TESIS DEFENDIDA POR

Irma Alejandra Amaya Patrón

Y APROBADA POR EL SIGUIENTE COMITÉ

Dr. Jesús Favela Vara
Director del Comité

M. C. Josefina Rodríguez Jacobo
Miembro del Comité

Dr. Pedro Negrete Regagnon
Miembro del Comité

M. C. Raúl Tamayo Fernández
Miembro del Comité

Dr. Gilberto López Mariscal
Coordinador del programa de posgrado en Ciencias de la Computación

Dr. Raúl Ramón Castro Escamilla
Director de Estudios de Posgrado

05 de Septiembre de 2005

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN SUPERIOR DE ENSENADA

PROGRAMA DE POSGRADO EN CIENCIAS EN CIENCIAS DE LA COMPUTACIÓN

COMPONENTES DE SOFTWARE PARA EL DESARROLLO DE APLICACIONES DE CÓMPUTO UBICUO

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de
MAESTRO EN CIENCIAS

Presenta:

IRMA ALEJANDRA AMAYA PATRÓN

Ensenada, Baja California, México. Septiembre de 2005.

RESUMEN de la tesis de **Irma Alejandra Amaya Patrón**, presentada como requisito parcial para la obtención del grado de **MAESTRO EN CIENCIAS** en **CIENCIAS DE LA COMPUTACIÓN**. Ensenada, Baja California. Septiembre de 2005.

Componentes de software para el desarrollo de aplicaciones de cómputo ubicuo

Resumen aprobado por:

_____

Dr. Jesús Favela Vara
Director de Tesis

La complejidad del desarrollo de aplicaciones de cómputo ubicuo (ubicomp) se debe en parte a la necesidad de considerar una variedad de aspectos tales como: comunicación, colaboración, movilidad, adaptabilidad, entre otros. Esfuerzos recientes orientados a agilizar y simplificar el desarrollo de aplicaciones ubicomp han adoptado nuevas técnicas y metodologías que incluyen elementos tales como: toolkits, frameworks, componentes, middlewares, de manera que éstas aplicaciones sean flexibles, robustas, adaptables y reutilizables. Dichos elementos proveen mayor nivel de abstracción puesto que pueden ser utilizados sin comprender a fondo la lógica de su implementación. Basándonos en esta idea, surge este trabajo, el cual consiste en crear un conjunto de componentes que faciliten al usuario el desarrollo de aplicaciones de cómputo ubicuo.

La base para desarrollar este trabajo es el middleware SALSA, el cual provee cierta funcionalidad limitada por el framework de clases que proporciona. En este trabajo se busca aumentar la funcionalidad de SALSA al proporcionar un conjunto de componentes que incrementen el nivel de granularidad de las unidades básicas de desarrollo y simplifiquen la tarea del desarrollador.

En particular, diseñamos e implementamos un componente que permite la transferencia de información entre diversos dispositivos que pueden ser heterogéneos. Este componente, denominado componente de migración, utiliza los mecanismos y servicios que provee el middleware SALSA para establecer la comunicación entre los distintos agentes que representan a los dispositivos. De esta forma, se permite transferir información de un dispositivo a otro, sin importar la naturaleza del mismo.

La evaluación del componente de migración se efectuó realizando la integración del mismo con otros componentes, en particular, con un componente de control remoto de dispositivos. Esto con la finalidad de ilustrar que el desarrollo basado en componentes permite la fácil y rápida construcción (integración) de aplicaciones y/o sistemas complejos.

Palabras clave: componentes de software, cómputo ubicuo, agentes, middleware.

ABSTRACT of the thesis presented by **Irma Alejandra Amaya Patrón** as a partial requirement to obtain the **MASTER OF SCIENCE** degree in **COMPUTER SCIENCE**. Ensenada, Baja California, Mexico. September 2005.

Software components to develop ubiquitous computing applications

Abstract approved by:

_____
Dr. Jesús Favela Vara
Thesis Director

The complexity of developing ubiquitous computing (ubicomp) applications is partly due to the need for designers to consider a variety of aspects dealing with communication, collaboration, mobility, and adaptability, among others. Recent efforts to facilitate the development of ubicomp applications have adopted new techniques and methodologies that include: toolkits, frameworks, components, middleware. These efforts aim to obtain applications that are flexible, robust, adaptable and reusable. These techniques provide a greater level of abstraction since they can be used without a thorough understanding of the logic of its implementation. Based on this idea, this proposes a set of components to facilitate the development of ubiquitous computing applications.
The components are developed based on the SALSA middleware and its class framework. As part of this work we look to increase SALSA's functionality by providing, through the use of components, units at a higher level of abstraction.
In particular, we design and implement a component that allows the transfer of information between heterogeneous devices. This component, called migration component, utilizes the mechanisms and services that provided by the SALSA middleware to establish the communication between the different agents that represent devices. Thus, allowing the transfer of information between devices, without concern as to their nature.
The evaluation of the migration component was conducted by integrating it with another component independently developed to remotely control devices. This is used to illustrate how component-based development allows for a fast and easy construction (integration) of applications and/or complex systems.

Keywords: software components, ubiquitous computing, agents, middleware.

Dedicatoria

A mi hijo.

Agradecimientos

A Dios,
por darme salud, y una hermosa familia.

Al Dr. Jesús Favela,
por haber sido un guía durante este tiempo.

A la Dra. Josefina Rodríguez, Dr. Pedro Negrete y al M. C. Raúl Tamayo,
por sus aportaciones para mejorar este trabajo.

A Norberto,
por todo su amor y comprensión.

A Simón,
por sus inocentes sonrisas que me impulsan a seguir adelante.

A mis compañeros,
por su amistad y ayuda.

A mis padres y hermanos,
por apoyarme en todo momento.

A la Dra. Edna Luna, M.C. Carmen Pérez y Dra. Ana Martínez,
por su apoyo y consejos.

A la M.I.S. Judith Luna,
por su gran apoyo y confianza.

Al Consejo Nacional de Ciencia y Tecnología.

CONTENIDO

Página

CONTENIDO (continuación)

Página

CONTENIDO (continuación)

Página

CONTENIDO (continuación)

Página

CONTENIDO (continuación)

Página

LISTA DE FIGURAS

LISTA DE FIGURAS (continuación)

LISTA DE FIGURAS (continuación)

LISTA DE FIGURAS (continuación)

LISTA DE FIGURAS (continuación)

LISTA DE TABLAS

Capítulo I

Introducción

I.1 Antecedentes

A través de los años, el desarrollo de sistemas de software ha evolucionado conforme han surgido nuevas técnicas y metodologías de desarrollo. La evolución de estas técnicas y metodologías ha requerido que los desarrolladores de sistemas tengan que adaptarse a los cambios que demanda dicha evolución. Esto con la finalidad de incrementar tanto el potencial de reusabilidad de las aplicaciones, como el nivel de abstracción de los elementos que forman parte de un sistema.

Los orígenes del desarrollo de sistemas se remontan al uso de lenguajes estructurados para crear aplicaciones de cómputo. Estos lenguajes evolucionan para dar vida a los lenguajes orientados a objetos, los cuales permiten llevar a cabo la programación orientada a objetos. La programación orientada a objetos permite aumentar el nivel de reusabilidad al proveer al desarrollador de librerías de clases que permiten la reutilización de su implementación. Sin embargo, los sistemas de cómputo actuales requieran más que la reutilización de código para agilizar su construcción. Debido a esto, surgen el diseño de patrones, los toolkits, las librerías de clases (frameworks), entre otros, que fomentan la reutilización del diseño. Finalmente, surgen los componentes de software que permiten la reusabilidad tanto de diseño como de implementación [Barr, 1999].

Los componentes de software proveen un enfoque de desarrollo donde se permite la construcción de aplicaciones o sistemas en base al ensamble de partes pre-fabricadas. El desarrollo de sistemas basado en componentes incrementa la productividad del

desarrollador, ya que reduce los tiempos de desarrollo, puesto que la construcción se basa en el diseño y la implementación que provee el componente.

Particularmente, este trabajo se enfoca a la construcción de componentes de software para el desarrollo de aplicaciones de cómputo ubicuo.

I.1.1 Cómputo Ubicuo

El Cómputo Ubicuo (ubicomp) representa el concepto de computación por todas partes, haciendo esencialmente el cómputo y la comunicación transparentes al usuario. Las características principales del ubicomp son [Abowd, 1999]:

- *Interacción transparente*: la visión del ubicomp es la computación pervasiva sin intrusión. Lo cual pretende eliminar las interfaces físicas como una barrera entre el usuario y el trabajo que se desea lograr a través de la computadora.
- *Captura automatizada*: con esto se busca la captura de experiencias diarias y llevar los registros para su uso posterior, a través de una herramienta que de soporte a la captura, integración y acceso a diversos tipos de información multimedia.
- *Consciente del contexto*: las aplicaciones conscientes del contexto son aquellas que permiten la recolección del contexto y el desempeño dinámico del programa guiado por el conocimiento del ambiente.

I.1.2 Componentes de software

Un componente de software es una unidad de composición de aplicaciones de software, que posee un conjunto de interfaces y satisface un conjunto de requisitos, y que puede ser desarrollado, adquirido e incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio. Además, un componente permite el desarrollo de aplicaciones a un mayor nivel de abstracción.

En un sistema ubicuo, los componentes deben interoperar de forma espontánea en ambientes cambiantes. Un componente interopera espontáneamente si este interactúa con

un conjunto de componentes que pueden cambiar su identidad y funcionalidad sobre el tiempo conforme cambia el ambiente de ejecución [Kindberg y Fox, 2002].

I.2 Planteamiento del Problema

El modelado de aplicaciones para cómputo ubicuo requiere tomar en consideración varios aspectos fundamentales, tales como la comunicación, conciencia del contexto, entre otros. Debido a esto, se adoptó el enfoque de: utilizar un conjunto suficientemente poderoso de abstracciones, para modelar partes significativas del mundo real. En otras palabras, el mundo que está siendo modelado en aplicaciones para cómputo ubicuo puede estar sujeto a cambios en el ambiente que nos rodea [Kindberg y Fox, 2002].

Dada la complejidad que presenta el desarrollo de aplicaciones para ambientes de cómputo ubicuo, surge la necesidad de proporcionar una herramienta que facilite la tarea de los desarrolladores. Una herramienta de este tipo debe tomar en consideración los principales retos de las aplicaciones, sistema operativo o middleware, y el lenguaje de desarrollo [Kindberg y Fox, 2002].

De esta forma, la herramienta estará formada por un conjunto de componentes, los cuales podrán ser utilizados sin mayor necesidad de comprender a fondo cual es la infraestructura bajo la cual se desarrollan. Simplemente, los desarrolladores deberán conocer cual es el dominio de la aplicación al cual está orientado el desarrollo de la aplicación ubicua.

Basándose en esto, para elaborar este trabajo se toma como referencia el middleware SALSA, el cual presenta una arquitectura bien definida consistente de varios módulos que permiten la comunicación, manejo e interacción entre los diversos agentes que forman parte de una aplicación consciente del contexto. Particularmente, SALSA ha sido usado para el desarrollo de aplicaciones médicas utilizando pantallas públicas y computadoras de bolsillo.

La arquitectura de SALSA se compone de los siguientes módulos [Rodríguez et al., 2005]:

- *Agent Broker,* es el encargado de manejar la comunicación entre agentes, los cuales representan usuarios, servicios y dispositivos.
- *SALSA Services,* es un conjunto de servicios los cuales permiten a los programadores registrar e inicializar los agentes en un directorio de agentes.
- *SALSA Class Framework,* provee un conjunto de clases que facilita el uso de dichos servicios y la implementación de un agente.

Puesto que la funcionalidad de SALSA reside en los servicios que proporciona el framework de clases, el cual está implementado a un nivel de abstracción relativamente bajo, la finalidad de este trabajo es incrementar el nivel de granularidad utilizando un conjunto de componentes. Dichos componentes deben proporcionar servicios o recursos genéricos que sirvan como base para el desarrollo fácil y eficiente de aplicaciones ubicuas, las cuales puedan desarrollarse utilizando el middleware SALSA y el sistema de componentes.

A partir de esta unión, podemos definir las principales características que ofrece COSSA (COmponentes de Software basados en SAlsa): flexibilidad, escalabilidad, mayor nivel de granularidad en el desarrollo de aplicaciones ubicuas, así como dinamismo en la evolución de las características que conforman a los componentes.

I.3 Objetivos

I.3.1 Objetivo General

Desarrollar un sistema de componentes que incrementen el alcance de SALSA y permitan el fácil desarrollo de aplicaciones para cómputo ubicuo.

I.3.2 Objetivos Específicos

Para llevar a cabo el objetivo general es necesario cumplir con los objetivos específicos que se presentan a continuación:

- Identificar componentes de software para ambientes de cómputo ubicuo

- Desarrollo de componentes
- Ampliar el alcance de SALSA
- Desarrollo de aplicaciones que utilicen los componentes desarrollados

I.4 Metodología

En la figura 1, se ilustra la metodología propuesta para el desarrollo de COSSA:

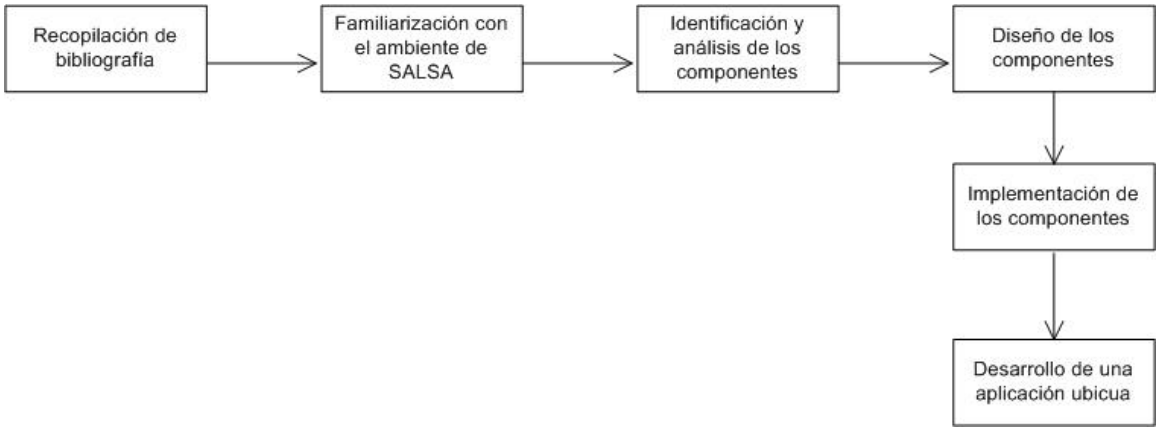


Figura 1. Metodología propuesta para el desarrollo de COSSA

Los elementos que componen dicha metodología son descritos a continuación:

- *Recopilación de bibliografía*: en esta etapa se busca recabar información de utilidad relacionada con el desarrollo de aplicaciones para cómputo ubicuo.
- *Familiarización con el ambiente de SALSA:* puesto que el desarrollo de esta investigación toma como base el trabajo realizado en SALSA, es necesario familiarizarse tanto con la información correspondiente a dicho trabajo, como con la implementación realizada.
- *Identificación y análisis de los componentes:* una vez realizado el estudio de la literatura, es necesario identificar que componentes van a desarrollarse para, posteriormente analizarlos y ver si cumplen con la finalidad de proporcionar funcionalidad genérica que incremente el alcance de SALSA.

- *Diseño de los componentes:* ya identificados los componentes que van a desarrollarse, se procede a diseñarlos.
- *Implementación de los componentes:* en esta etapa se procede a implementar los servicios o recursos que proporcionan los componentes.
- *Desarrollar una aplicación ubicua:* utilizando los componentes desarrollados se pretende elaborar una aplicación que los utilice y demuestre su funcionalidad.

I.5 Organización de la Tesis

El trabajo desarrollado en esta tesis consta de siete capítulos y varios apéndices. A continuación, se presenta una breve descripción de cada uno de ellos.

El *capítulo II* presenta los conceptos y definiciones de los componentes de software y sus elementos. En este capítulo se mencionan también las ventajas y desventajas del desarrollo basado en componentes, las distintas tecnologías de componentes existentes, y las diferencias entre objetos y componentes. Finalmente, se presenta la metodología de desarrollo a seguir en este trabajo para la construcción de componentes.

En el *capítulo III* se presenta una breve introducción al cómputo ubicuo. Además, se realiza un análisis del trabajo relacionado en esa área donde se presentan algunos escenarios de uso. Finalmente, se realiza la identificación y descripción de componentes de software identificados en dichos escenarios, para concluir con la descripción de la relación entre middleware y componentes.

El *capítulo IV* presenta el análisis del componente de migración. Este capítulo incluye la descripción de ejemplos de aplicaciones, la elaboración de escenarios detallados, identificación de los servicios asociados al componente y la especificación de requerimientos. Por último, se presenta la descripción de los casos de uso del componente de migración.

En el *capítulo V* se describe el diseño y la implementación del componente de migración. Este capítulo incluye la arquitectura del sistema, los diagramas de secuencia, así como el

modelo del componente y diagramas de clase. Finalmente, se presenta el prototipo del componente, así como ejemplos de pantallas que ilustran la funcionalidad que este ofrece.

El *capítulo VI* presenta el caso de estudio que describe la implementación de un ambiente de cómputo ubicuo basado en componentes situado en un contexto hospitalario. En este capítulo, se ilustran también las distintas formas de interacción que pueden presentarse entre los diversos componentes.

Finalmente, en el *capítulo VII* se presentan las conclusiones de este trabajo de tesis, aportaciones y propuestas de trabajo futuro.

Capítulo II

Componentes de Software

II.1 Introducción

El desarrollo de sistemas de cómputo ha evolucionado conforme el paso de los años. Actualmente, la tendencia es incrementar la productividad del desarrollador. Es por esto, que se han propuesto nuevos elementos o técnicas que provean un mayor nivel de abstracción. Con esto, se busca aumentar el potencial de reutilización de los sistemas. El conjunto de estos dos elementos permite al desarrollador crear las aplicaciones en un menor tiempo. En la figura 2, se muestra de forma gráfica como han ido evolucionando las distintas tecnologías que son utilizadas en el desarrollo de sistemas [Barr, 1999].

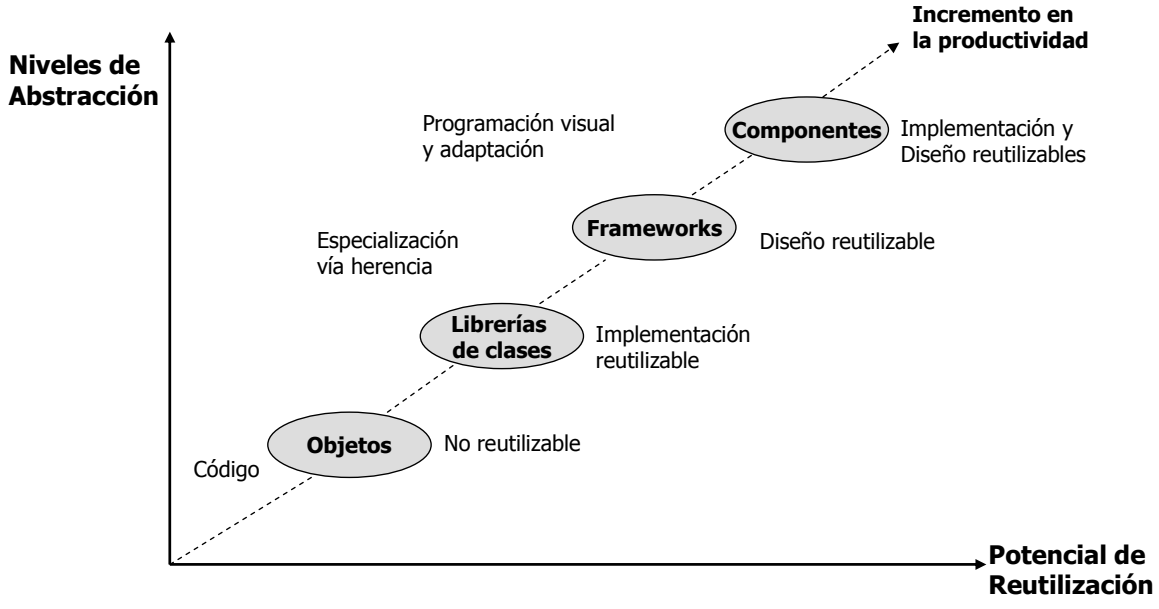


Figura 2. Evolución de las tecnologías utilizadas en el desarrollo de sistemas

La evolución de las tecnologías inicia a partir de la aparición de los objetos, los cuales no podían ser reutilizados en otras aplicaciones. Para superar esta limitante, surgieron las librerías de clases, las cuales proveen una implementación reutilizable. Sin embargo, los elementos que la componen pueden ser heredados de otras clases. Los frameworks proporcionan una estructura que facilita el diseño reutilizable, a través de un lenguaje de programación visual. Tomando estas dos últimas tecnologías surgen los componentes de software, los cuales permiten la reutilización de la implementación y el diseño.

Un componente de software es una unidad de composición de aplicaciones de software, que posee un conjunto de interfaces y satisface un conjunto de requisitos, y que puede ser desarrollado, adquirido e incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio. Además, un componente permite el desarrollo de aplicaciones a un mayor nivel de abstracción.

En un sistema ubicuo, los componentes deben interoperar de forma espontánea en ambientes cambiantes. Un componente que interactúa con su ambiente debe ser capaz de cambiar sin necesidad de nuevo software o parámetros adicionales [Kindberg y Fox, 2002].

La utilización de componentes en el desarrollo de sistemas ofrece varias ventajas [Griss y Pour, 2001; McInnis, 2000]:

- Reducción de costo y tiempo de desarrollo del sistema, al permitir al desarrollador ensamblar dichos sistemas partiendo de componentes reutilizables.
- Incrementar la confiabilidad de los sistemas, puesto que cada componente reutilizable es revisado, inspeccionado y aprobado para proporcionar un correcto desempeño en el sistema.
- Mejorar el mantenimiento de los sistemas, ya que permite incluir nuevos componentes, mejorar la calidad de los que ya existen y reemplazar a aquellos que se requiera.
- Incrementar la calidad de los sistemas, ya que se tienen a expertos en el dominio de la aplicación trabajando en la infraestructura. De esta forma, se tiene una estructura

base bien definida, la cual permite a otros usuarios tomar los componentes de la misma y desarrollar la aplicación.

Las siguientes secciones, definen que es un componente de software, cuales son sus elementos, así como las ventajas y desventajas que proporciona el desarrollo de software basado en componentes.

II.2 Componentes

En un ambiente o entorno, existen componentes los cuales son unidades de software que implementan abstracciones tales como servicios, clientes, recursos u otras aplicaciones [Kindberg y Fox, 2002]. Dicho de otra forma, un componente es una unidad de desempeño independiente que interactúa con su ambiente a través de interfaces bien definidas las cuales encapsulan su implementación [Griss y Pour, 2001].

[Szyperski et al., 2002] define un componente de software como:

"Un componente de software es una unidad de composición con interfaces especificadas contractualmente y dependencias explicitas de contexto. Un componente de software puede ser ejecutado independientemente y es sujeto de composición con terceros".

En base a esta definición, se pueden identificar los siguientes aspectos:

- Interfaces bien definidas.
- Dependencias explícitas del contexto.
- Pueden ser ejecutados independientemente.
- Composición con terceros.

Estos aspectos tienen varias implicaciones. Para que un componente pueda ser *ejecutado independientemente*, necesita tener una clara separación de su ambiente y el de otros componentes. Por lo tanto, un componente encapsula las características que lo constituyen. Como unidad ejecución, el componente no puede ser ejecutado parcialmente. En cuanto a la *composición con terceros,* el componente necesita ser lo suficientemente autocontenido.

Además, necesita estar acompañado de una clara especificación de que es lo qué requiere y qué es lo que proporciona. En otras palabras, un componente necesita encapsular su implementación e interactuar con su ambiente a través de las interfaces. Las *interfaces bien definidas* representan los únicos puntos de acceso mediante los cuales los usuarios pueden acceder a los servicios que ofrece el componente. Las *dependencias del contexto* se refieren a las especificaciones que requiere el ambiente de ejecución para que el componente pueda funcionar [Szyperski et al., 2002].

Además, los componentes pueden ser vistos como agentes, si estos exhiben una combinación de las siguientes características (ver figura 3) [Gris, 2001]:

- *Autonomía*: el agente debe iniciar proactivamente actividades relacionadas con su propósito, deben poseer su propio control, y pueden actuar por su cuenta, independientemente de los mensajes que envíen otros agentes.
- *Adaptabilidad:* el agente debe ser capaz de aprender, adaptar o adquirir nuevas capacidades para cambiar su desempeño.
- *Conocimiento:* el agente puede razonar acerca de sus propósitos, adquiriendo información y conocimiento de otros agentes y usuarios.
- *Movilidad:* el agente puede moverse de un contexto de ejecución a otro, moviendo su código y empezando de nuevo o serializando el código y el estado para continuar su ejecución en un nuevo contexto y manteniendo su estado para continuar su trabajo.
- *Colaboración:* el agente puede comunicarse y trabajar cooperando con otros agentes de forma dinámica o estática colaborando conjuntamente para realizar una tarea.
- *Persistencia:* la infraestructura permite a los agentes conservar conocimiento y estado por largos períodos de tiempo.

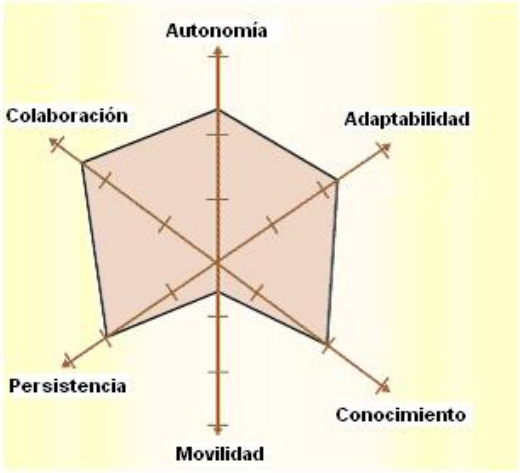


Figura 3. Características de los componentes para ser denominados agentes [Griss, 2001]

En la siguiente sección, se describen los elementos que integran a un componente de software.

II.3 Elementos de los componentes

Un componente de software está formado de varios elementos (ver figura 4):

- Interfaz.
- Receptáculos.
- Atributos.
- Depósitos y fuentes de eventos

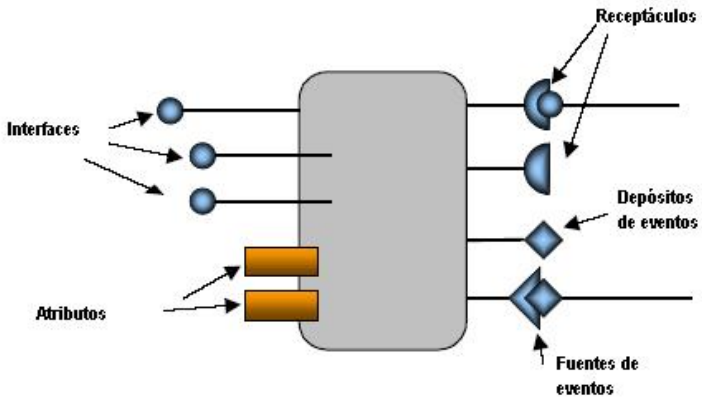


Figura 4. Componente de software y sus elementos

La especificación de cada uno de estos elementos describe el modelo del componente. El conjunto de dichos elementos define al componente como tal. A continuación, en las siguientes secciones se realiza la definición de cada uno de estos elementos.

II.3.1 Interfaces

Una interfaz es una colección de operaciones que definen cierta funcionalidad. Las interfaces son mecanismos a través de los cuales los componentes describen los servicios que proporcionan [Supriya, 2002].

La interfaz de un componente puede ser definida como una especificación de sus puntos de acceso, a través de los cuales los clientes acceden a los servicios que provee el componente. La interfaz no ofrece implementación de sus operaciones. Sólo nombra una colección de operaciones y proporciona las descripciones y protocolos de dichas operaciones. Esto permite: reemplazar parte de la implementación sin cambiar la interfaz, y de esta forma mejorar el desempeño del sistema sin reconstruirlo; y agregar nuevas interfaces (e implementaciones) sin cambiar la implementación existente, y de esta forma mejorar la adaptabilidad del componente [Crnkovic et al, 2002].

II.3.2 Receptáculos

Los receptáculos son unidades que requieren un servicio. Indican una dependencia explícita entre componentes. Describen una posible conectividad, la habilidad para especializarse por delegación y la habilidad para componer funciones del componente. Son puntos de acceso para interfaces de otros componentes, y proporcionan un medio para que otros componentes puedan conectarse a él [Boertien et al, 2001].

II.3.3 Atributos

Los atributos son valores de los componentes utilizados para configurarlos en la etapa de diseño o al momento de ejecución [Boertien et al, 2001].

II.3.4 Depósitos y fuentes de eventos

Los depósitos y fuentes de eventos especifican la forma en la que el componente notifica al exterior una respuesta a un estímulo externo o bien un cambio en una condición interna.

En las siguientes secciones, se citan las ventajas y desventajas del desarrollo de software basado en componentes.

II.4 Ventajas de los componentes

El desarrollo de sistemas basado en componentes de software ofrece varias ventajas al desarrollador, tales como [Supriya, 2002; McInnis, 2000]:

- *Mejor calidad*: las partes utilizadas en múltiples ocasiones poseen menos defectos que aquellas que han sido fabricadas recientemente.
- *Menores riesgos*: el riesgo en la creación de nuevos sistemas de software se reduce utilizando componentes que ya encapsulan la funcionalidad deseada.
- *Menor tiempo de desarrollo*: el desarrollo basado en componentes permite al desarrollador enfocarse en los problemas de la aplicación, en lugar de los detalles de bajo nivel de programación.
- *Facilidad de desarrollo:* las aplicaciones pueden ser desarrolladas más fácilmente al tomar los componentes ya existentes para adecuarlos y ensamblarlos en sus aplicaciones.
- *Incremento en la productividad:* dado que el análisis y diseño de la aplicación requiere menor tiempo.
- *Soporte a prototipos:* la disponibilidad de componentes reutilizables da soporte a prototipos y a la validación de requerimientos del usuario.
- *Menores costos*: el desarrollo de componentes puede ser más barato que el desarrollo tradicional, dado que los costos de desarrollo y mantenimiento se ven reducidos con la utilización de componentes.

- *Captura de habilidades técnicas y conocimiento*: la experiencia de los desarrolladores es capturada en componentes reutilizables que pueden ser usados por personal menos experimentado.
- *Flexibilidad*: en la selección de herramientas y lenguajes a utilizar para implementar la solución como componentes interoperables.
- *Entrada a nuevos segmentos del mercado:* cuando los sistemas de software pueden ser construidos usando o (reutilizando) componentes que posiblemente fueron elaborados por distintas compañías, se abren las puertas para la compra-venta de componentes con la finalidad de desarrollar componentes más complejos, o bien, adecuarlos a las necesidades propias.

Sin embargo, no obstante las ventajas que ofrece el desarrollo basado en componentes, existen ciertas desventajas y riesgos, los cuales son descritos a continuación.

II.5 Desventajas y riesgos de los componentes

El desarrollo de sistemas basados en componentes implica ciertas desventajas y riesgos que deben ser analizados por el desarrollador para considerar su uso en el desarrollo de un sistema. Entre las desventajas y riesgos podemos citar [Crnkovic, 2001]:

- *Tiempo y esfuerzo requerido para el desarrollo de componentes*: la construcción de unidades reutilizables requiere de 3 a 5 veces más esfuerzo que el requerido para desarrollar una unidad para un propósito específico.
- *Requerimientos ambiguos y poco claros*: puesto que los componentes son reutilizables, por definición, estos pueden ser usados en diferentes aplicaciones, algunas de las cuales pueden ser desconocidas y los requerimientos no pueden ser predecidos al momento de su diseño y desarrollo.
- *Conflictos entre usabilidad y reusabilidad*: para ser suficientemente reutilizable, un componente debe ser lo suficientemente general, escalable y adaptable, y por lo tanto más complejo (mayor complejidad en el uso) y demandante de recursos de cómputo (más caro en el uso).

- *Costos de mantenimiento del componente*: los costos de mantenimiento del componente pueden ser muy altos debido a que el componente debe responder a diferentes requerimientos de distintas aplicaciones ejecutándose en distintos ambientes, con diversos requerimientos de confiabilidad y quizás requiera de un nivel diferente de soporte al mantenimiento.
- *Confiabilidad y sensibilidad a cambios*: dado que, los componentes y las aplicaciones poseen ciclos de vida separados y satisfacen distintos tipos de requerimientos, existen riesgos de que un componente no pueda satisfacer completamente los requerimientos de la aplicación o que estos puedan incluir características encubiertas no conocidas para los desarrolladores de la aplicación. Cuando se introducen cambios a nivel aplicación, existe un riesgo de que el cambio introducido pueda ocasionar alguna falla en el sistema.

En base a estos riesgos y desventajas el desarrollador puede tomar la decisión de implementar o no su sistema utilizando componentes. A continuación, se describen algunas tecnologías comerciales de componentes de software.

II.6 Tecnologías de componentes

Algunas tecnologías, tales como Visual Basic Controls, Controles ActiveX, librerías de clases, y JavaBeans, hacen posible que sus lenguajes relacionados (Visual Basic, C++, Java) soporten herramientas para compartir y distribuir piezas de la aplicación. Sin embargo, el funcionamiento de todos estos enfoques recae en ciertos servicios necesarios para establecer la comunicación y coordinación en la aplicación. En base a esto, las tecnologías de componentes actúan como un puente que permite la comunicación entre los componentes [Cai et al., 2000].

Entre las diversas tecnologías de componentes que han sido desarrolladas, existen tres que han sido estandarizadas: CORBA, COM/DCOM, y JavaBeans. La estandarización de dichas tecnologías ha permitido que sean utilizadas como base para el desarrollo de distintas aplicaciones y sistemas. A continuación, se describe cada una de estas tecnologías.

II.6.1 Common Object Request Broker Architecture (CORBA)

CORBA es un estándar abierto para la interoperabilidad de aplicaciones definido y soportado por el OMG (Object Management Group). CORBA maneja los detalles de interoperabilidad del componente, y permite a las aplicaciones comunicarse unas con otras en forma remota. La única forma de comunicación entre las aplicaciones o componentes es la interfaz. La parte más importante de CORBA es el ORB (Object Request Broker). El ORB es el middleware que establece las relaciones cliente-servidor entre los componentes. Utilizando un ORB, un cliente puede invocar un método en un servidor de objetos, cuya ubicación es completamente transparente al usuario. El ORB es el responsable de interceptar una llamada y buscar un objeto que implemente la solicitud, pasando los parámetros, invocando sus métodos, y regresando los resultados [Cai et al., 2000; Szyperski et al., 2002].

En un modelo de componente de CORBA, el cliente no necesita conocer donde está ubicado el objeto, el lenguaje en el que fue programado, el sistema operativo sobre el que se ejecuta, o cualquier característica adicional que no se encuentre disponible a través de la interfaz. De esta forma, el ORB proporciona interoperabilidad entre aplicaciones en distintas máquinas en ambientes distribuidos heterogéneos. CORBA es ampliamente utilizada en sistemas orientados a objetos distribuidos incluyendo sistemas de software basados en componentes, puesto que ofrecen un ambiente consistente de distribuido de programación y ejecución sobre lenguajes comunes de programación, sistemas operativos, y redes distribuidas.

II.6.2 Component Object Model (COM) y Distributed Com (DCOM)

COM es una arquitectura general para componentes de software. Proporciona una plataforma dependiente, basada en Windows y aplicaciones basadas en componentes independientes del lenguaje [Cai et al., 2000; Szyperski et al., 2002].

COM define la forma en la cual el componente interactúa con sus clientes. Esta interacción esta definida de tal forma que el cliente y el componente pueden conectarse sin la necesidad de intermediarios. Especialmente, COM provee un estándar binario que deben seguir los

componentes y sus clientes para asegurar la interoperabilidad dinámica. Esto permite las actualizaciones automáticas del software y la reusabilidad.

Como una extensión del COM surge DCOM. DCOM es un protocolo que permite a los componentes de software comunicarse directamente sobre una red en forma confiable, segura y eficiente. DCOM esta diseñado para usarse sobre una variedad de redes, incluyendo protocolos de Internet, como el HTTP. Cuando un cliente y su componente residen en distintas maquinas, DCOM simplemente reemplaza el proceso de intercomunicación local por un protocolo de red. El cliente o el componente nunca están conscientes de los cambios físicos de la conexión.

II.6.3 JavaBeans y Enterprise JavaBeans (EJB)

Sun desarrolla un modelo de componente basado en Java que consiste de dos partes: JavaBeans como componente de desarrollo del cliente y Enterprise JavaBeans como componente de desarrollo del lado del servidor. La arquitectura de componentes JavaBeans soporta aplicaciones multi-plataformas, lo cual permite la reutilización de los componentes del cliente y servidor [Cai et al., 2000; Szyperski et al., 2002].

La plataforma de Java ofrece una solución eficiente a los problemas de portabilidad y seguridad, a través del uso de códigos portables de Java y conceptos de autentificación. Java provee una integración universal y habilita la tecnología para el desarrollo de aplicaciones, incluyendo: interoperatividad entre distintos servidores; propagación de transacciones y contextos seguros; soporte ActiveX a través de puentes DCOM/CORBA.

La portabilidad, seguridad, confiabilidad que ofrece Java permite el desarrollo de componentes robustos, independientes del sistema operativo, y del servidor.

II.6.4 Comparación de las tecnologías

En base a la información obtenida en los puntos anteriores, se realiza un análisis comparativo de las distintas tecnologías de componentes. En la Tabla I, se muestra la tabla comparativa entre las tecnologías de CORBA, EJB y COM/DCOM. En dicha tabla pueden apreciarse las características principales de dichas tecnologías.

Tabla I. Comparación de las tecnologías de componentes

	CORBA	EJB	COM/DCOM
Ambiente de desarrollo	Bajo desarrollado.	Emergente.	Soportado por un amplio rango de fuertes ambientes de desarrollo.
Estándar binario	No posee estándares binarios.	Basado en COM, Java específicamente.	Un estándar binario para la interacción entre componentes.
Compatibilidad y portabilidad	Fuerte, particularmente en asociaciones de estandarización del lenguaje, pero no es portable.	Portable en la especificación del lenguaje Java, pero no muy compatible.	No posee ningún concepto de estándar a nivel de código o de un estándar para la asociación del lenguaje.
Modificación y mantenimiento	El IDL de CORBA para la definición de interfaces. Requiere modificación y mantenimiento adicional.	No incluido en los archivos IDL, la definición de interfaces entre los componentes y el contenedor. Facilidad de modificación y mantenimiento.	El IDL de Microsoft para definir las interfaces de los componentes, requiere modificación y mantenimiento adicional.
Servicios que provee	Un conjunto completo de servicios estandarizados, carece de implementaciones.	Sin estándar.	Recientemente suplementado por un número de servicios claves.
Plataforma	Independencia de plataforma.	Independencia de plataforma.	Dependencia de plataforma.
Lenguaje	Independencia del lenguaje.	Dependiente del lenguaje.	Independiente del lenguaje.
Implementación	Fuertemente para el cómputo tradicional de empresas.	Fuertemente en clientes Web.	Fuertemente en aplicaciones tradicionales de escritorio.

II.7 Diferencias entre objetos y componentes

Un componente actúa a través de objetos y por lo tanto, consisten normalmente de una o más clases u objetos. De esta forma, el componente puede contener un conjunto de objetos inmutables que capturen por defecto el estado inicial y otros recursos del componente. Sin embargo, no existe la necesidad de que un componente contenga clases solamente. En su lugar, un componente puede contener procedimientos tradicionales y variables globales. Los objetos creados en un componente pueden abandonar el componente y volverse invisibles a los clientes del componente [Szyperski et al., 2002].

A partir de estas definiciones podemos observar que existe una clara diferencia entre los objetos y componentes. La Tabla II presenta las formas en las cuales difieren los objetos de los componentes [Supriya, 2002].

Tabla II. Diferencias entre objetos y componentes

Objetos	Componentes
▪ Limitados a los lenguajes orientados a objetos.	▪ Pueden ser escritos en cualquier lenguaje.
▪ Muchas dependencias firmes, tales como: implementación de la interfaz, fragilidad en las clases base.	▪ Menor aclopamiento que en los objetos.
▪ Unidades de composición de fina granularidad.	▪ Mayor granularidad que los objetos.
▪ Formas limitadas de conectarse (invocación de métodos).	▪ Soporte a múltiples interfaces y diseño orientado a la interfaz.
▪ Conjunto limitado de servicios a los cuales dan soporte: seguridad, transacciones, etc.	▪ Soporta más formas dinámicas de asociación y descubrimiento de servicios.
▪ Diseño casado con los principios orientado a objetos.	▪ Mejores mecanismos para la composición con terceros.
	▪ Más soporte para servicios de alto orden (seguridad, transacciones, etc.).
	▪ El diseño se apega a reglas relacionadas a frameworks de componentes.

Una vez definidas las diferencias entre objetos y componentes, se presenta la metodología de desarrollo basada en componentes en la siguiente sección.

II.8 Metodología de desarrollo

La metodología para el desarrollo basado en componentes (CBD - Component Based Development) comprende varias etapas en las cuales se incluyen una serie de técnicas e información útiles en el proceso de desarrollo.

En este caso, se adopta una combinación de las metodologías propuestas por [McInnis, 2000; Supriya, 2002]. Las etapas definidas en la metodología de desarrollo son:

Entender los requerimientos → Análisis → Definir la arquitectura → Implementar la solución → Probar la implementación.

A continuación, se definen las actividades que se realizan en cada una de estas etapas.

1) *Entender requerimientos*. Establecer los requerimientos del sistema es el punto inicial de la mayoría de los proyectos que involucran el desarrollo de aplicaciones. Los requerimientos usualmente se basan en decidir que procesos del negocio soportará el sistema. Inicialmente, estos pueden ser simples enunciados, los cuales, posteriormente son expandidos conforme se detalla el sistema. Una vez que los procesos del negocio han sido identificados, requerimientos adicionales al sistema se describen durante la etapa del análisis de desempeño. Las actividades que se realizan en esta etapa son las siguientes:
    - Obtener requerimientos.
    - Decidir el alcance de la aplicación.
    - Analizar los requerimientos.

2) *Análisis.* La etapa de análisis consiste en decidir la funcionalidad que ofrecerá el sistema. De esta forma, una vez que los requerimientos han sido establecidos, deben entenderse los detalles de cómo se realizan los procesos y en que forma el sistema dará soporte a estos. Esta etapa consta de las siguientes actividades:
    - Definición de escenarios.
    - Identificación de servicios.

- Definición de entradas y salidas.
- Elaboración de casos de uso.
- Definición de interacción con otros componentes.

3) *Definir la arquitectura.* La definición de la arquitectura consiste en la definición del conjunto de elementos que provee el componente. Así como la forma en que estos interaccionan. Las actividades requeridas en esta etapa son:
   - Identificar los elementos que forma parte de la solución.
   - Definir el modelo del componente.
   - Elaborar el diagrama de clases.
   - Elaborar diagramas de secuencia.

4) *Implementar la solución*. Implementar la solución consiste en desarrollar la lógica del componente (codificar). Los puntos que se realizan en esta etapa son:
   - Implementar los componentes.
   - Integrar y componer la aplicación.

5) *Probar la implementación*. Para probar la implementación se requiere:
   - Probar los componentes.
   - Probar el incremento de la aplicación compuesta desde los componentes.

II.9 Resumen

En este capítulo se presentaron conceptos básicos de los elementos de componentes de software. Así como las ventajas y desventajas del desarrollo basado en componentes. También se mostró un estudio de las distintas tecnologías de componentes, de donde, se resume dicho estudio en una tabla comparativa de las distintas tecnologías. Además, se definieron algunas diferencias entre objetos y componentes. Para finalizar, se definió la metodología de desarrollo a utilizar en este trabajo.

En el siguiente capítulo, se realiza el análisis y definición de componentes de software para cómputo ubicuo.

Capítulo III

Componentes de software para cómputo ubicuo

III.1 Introducción

A partir de la visión de Mark Weiser, quien introduce la idea de cómputo ubicuo o ubicomp, donde la computación se mueve fuera del escritorio y se integra al ambiente que nos rodea, surge la necesidad de desarrollar aplicaciones que involucren la participación de componentes del mundo real, los cuales deban ajustarse a ambientes y contextos de uso particulares [Dey et al., 2004; Kindberg y Fox, 2002].

Las aplicaciones de este tipo con frecuencia son conscientes del contexto, lo cual significa que puedan adaptarse dinámicamente a cambios en las actividades y ambientes del usuario, donde la comunicación e interacción sea transparente para el usuario [Dey et al., 2004].

La búsqueda constante de aplicaciones flexibles, adaptables, extensibles y robustas que puedan desarrollarse de manera rápida y relativamente sencilla requiere de nuevas metodologías y estrategias de desarrollo [Griss y Pour, 2001]. Particularmente, en el desarrollo de algunos sistemas ubicuos se han propuesto nuevos elementos que simplifican y agilizan la elaboración de aplicaciones de este tipo.

Estos elementos, tales como: middleware, componentes, toolkits, frameworks, y lenguajes visuales, permiten al desarrollador elaborar aplicaciones a partir de servicios y recursos básicos que pueden adaptarse fácilmente al contexto de la aplicación. La inclusión de dichos elementos en el desarrollo de sistemas ubicuos provee ciertas ventajas, tales como [Griss y Pour, 2001]:

- Reducción de costo y tiempo de desarrollo del sistema, al permitir al desarrollador ensamblar dichos sistemas partiendo de componentes reutilizables.
- Incrementar la confiabilidad de los sistemas, puesto que cada componente reutilizable es revisado, inspeccionado y aprobado para proporcionar un correcto desempeño.
- Mejorar el mantenimiento de los sistemas, ya que permite incluir nuevos componentes, mejorar la calidad de los ya existen y reemplazar a aquellos que se requiera.
- Incrementar la calidad de los sistemas, ya que se tienen a expertos en el dominio de la aplicación trabajando en la infraestructura.

Tomando en cuenta estas ventajas, el enfoque del desarrollo de sistemas ubicuos se dirige a la aplicación de estos elementos como parte de un todo. Donde el middleware proporcione servicios genéricos que permitan interactuar fácil y eficientemente a los diversos componentes de una aplicación [Bellifemine et al., 2003; Kindberg y Fox, 2002]; los componentes sean unidades de software que implementen abstracciones tales como servicios, clientes, recursos u otras aplicaciones [Kindberg y Fox, 2002]; y el lenguaje de programación visual sea el encargado de tomar todos los componentes y representarlos gráficamente para que los desarrolladores puedan establecer las normas básicas de comunicación e interacción.

A partir de este enfoque, se busca el desarrollo de un sistema de componentes que permita la integración de dichos elementos para desarrollar sistemas ubicuos tomando una funcionalidad base (provista por el middleware) y nuevos componentes (a desarrollar), los cuales podrían integrarse en un lenguaje visual para facilitar y simplificar la elaboración de sistemas. Permitiendo de esta forma al desarrollador tomar los componentes, relacionarlos y establecer las normas de comunicación para obtener cierta funcionalidad. En las siguientes secciones se detalla la búsqueda de dicho sistema de componentes.

III.2 Introducción al cómputo ubicuo

El cómputo ubicuo se refiere al diseño de ambientes saturados con computación y comunicación inalámbrica naturalmente integrados en las actividades humanas

[Satyanarayanan, 2002]. A partir de la definición de este concepto Mark Weiser establece las etapas u olas en las que divide la historia de la computación (ver figura 5). En la primer ola aparecieron los mainframes, donde existía una sola computadora para muchos usuarios. La segunda ola, o era de la computación personal, es donde una sola persona utiliza una computadora. La tercer ola hace referencia al cómputo ubicuo, lo cual involucra que un usuario pueda utilizar varias computadoras, de tal forma que el usuario no percibe la presencia de las máquinas. Esta característica permite al usuario enfocarse en sus tareas y en interactuar con otros usuarios [Weiser, 1996].

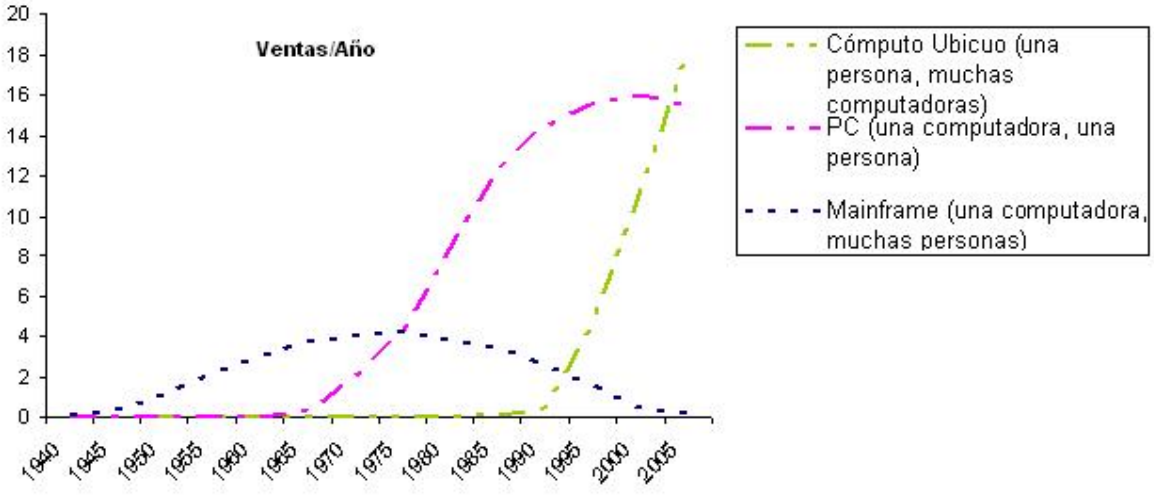


Figura 5. Olas de la computación [Weiser, 1996].

Actualmente, la tercer ola de la computación se encuentra en plena etapa de realización dado el creciente interés en el desarrollo de sistemas ubicomp. En este tipo de sistemas se plantea la utilización de grandes números de dispositivos y sensores embebidos en el ambiente. Debido a la demanda y las nuevas tecnologías emergentes que son utilizadas para el desarrollo de sistemas, los costos de los dispositivos y sensores han disminuido [Gilleade et al., 2003]. Con esta reducción de costos, se facilita el desarrollo de sistemas ubicomp, puesto que la tecnología ya no es considerada una limitante por sus altos costos.

Dentro de las principales características que provee un sistema ubicomp tenemos [Kindberg y Fox, 2002]:

- *Interacción transparente*: busca la eliminación de interfaces físicas como una barrera entre el usuario y el trabajo que se desea lograr a través de la computadora.
- *Captura Automatizada*: significa la recopilación de experiencias diarias, a través de una herramienta que de soporte a la captura, integración y acceso a diversos tipos de información.
- *Consciente del contexto*: las aplicaciones conscientes del contexto son aquellas que permiten la recolección del contexto y desempeño dinámico del programa guiado por el conocimiento del ambiente.

En la siguiente sección, se presenta una recopilación del trabajo relacionado con el área de cómputo ubicuo. Dicho trabajo sirve como base para el análisis e identificación de componentes de software.

III.2.1 Trabajo relacionado

Los siguientes proyectos muestran el trabajo de investigación relacionado con el desarrollo de aplicaciones ubicomp.

III.2.1.1 Phidgets

Phidgets [Greenberg, 2004], es una herramienta que utiliza un toolkit de componentes que permite al desarrollador elaborar fácilmente interfaces gráficas mediante el uso de widgets (componentes gráficos tales como: botones, cajas de texto, menús, etc). *Phidgets* utiliza componentes denominados phidgets los cuales permiten: coleccionar datos referentes a cambios en el ambiente, darles forma y acoplarlos en el modelo que contiene la información relacionada con el sistema. De esta forma, se permite al desarrollador crear aplicaciones tomando componentes e integrarlos en una interfaz para definir sus características y funcionalidad (ver figura 6).

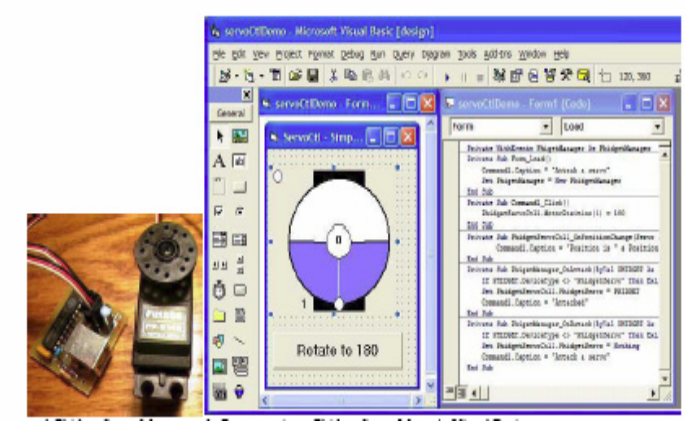


Figura 6. Interfaz de Phidgets.

Esta herramienta se encuentra aún en desarrollo puesto que, se están analizando nuevas formas de colaboración en las que puedan aplicarse los componentes. Dichos componentes representan dispositivos físicos, los cuales buscan integrarse dentro de ambientes reales donde exista un mayor nivel de colaboración con otros componentes, tales como: pantallas públicas, ambientes compartidos, etc.

III.2.1.2 a CAPpella

a CAPpella [Dey et al., 2004] presenta un prototipo de ambiente consciente del contexto, el cual permite al usuario final construir aplicaciones sin la necesidad de escribir código alguno. *a CAPpella* utiliza una combinación del aprendizaje de máquina y entradas del usuario para dar soporte a la construcción de aplicaciones conscientes del contexto a través de la programación por demostración. La idea principal que aporta *a CAPpella* es no requerir usuarios finales con experiencia en la creación de componentes, en su lugar, la herramienta crea los componentes por ellos, permitiéndole al usuario entender y mejorar su capacidad de interpretación del comportamiento de los componentes (ver figura 7).

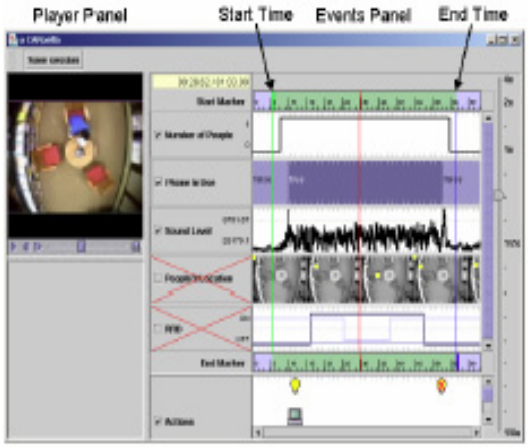


Figura 7. Interfaz de *a Capella.*

Esta herramienta presenta ciertas desventajas: requiere un entrenamiento previo del comportamiento que ofrecen los componentes, para lo cual necesita un amplio número de ejemplos y datos como referencia; no se ha probado la escalabilidad del sistema, puesto que no se han hecho pruebas para verificar el soporte a múltiples usuarios en diversos lugares con más de un componente; el número de escenarios de prueba es limitado, por lo cual se requiere un estudio para evaluar nuevos posibles escenarios de uso.

III.2.2 Escenarios ubicomp

En los siguientes escenarios se presentan algunas situaciones donde puede apreciarse el concepto de ubicomp. Dichos escenarios se presentan en un ambiente hospitalario, donde interactúan médicos y enfermeras para analizar o tomar decisiones sobre la salud de los pacientes. En base a la información que proveen estos escenarios, se realiza la identificación de algunos componentes de software.

Escenario 1. Se desea localizar una pantalla pública para proyectar una presentación

El Dr. Pérez desea proyectar su presentación en una pantalla pública. El archivo de la presentación se encuentra en su PDA, por esta razón, necesita conocer la ubicación de la pantalla pública más cercana. Para esto, el Dr. Pérez consulta en su PDA el mapa que muestra la localización de los dispositivos. El Dr. Pérez observa en el mapa cual(es) pantalla pública es la más cercana a su ubicación. Una vez que el Dr. Pérez ha localizado la ubicación de la pantalla se dirige hacia ésta. La pantalla pública más cercana se encuentra

ubicada en el salón de conferencias, y por esta razón, el Dr. Pérez se dirige hacia allá. Ya en el salón de conferencias, el Dr. Pérez desea transferir la información del archivo de la presentación de su PDA a la pantalla pública que se encuentra ahí. El Dr. Pérez transfiere la información de su PDA a la pantalla para que de esta forma, sus colegas puedan observar la presentación en la pantalla pública. Para iniciar con la presentación, el Dr. Pérez decide controlar la presentación desplegada en la pantalla pública a través de su PDA, ya que esto le permite efectuar la presentación independientemente de la posición en la que se encuentre en el salón.

A partir de este escenario se pueden identificar tres componentes: localización de servicios, migración de información y control remoto de dispositivos.

Escenario 2. Se busca localizar un usuario para colaborar con él

La Dra. García desea conocer la ubicación de la Dra. López para analizar los resultados de un paciente. Para esto, la Dra. López se encuentra parada enfrente de una pantalla pública. La Dra. García consulta en su PDA el mapa para determinar la ubicación de la Dra. López. Una vez que conoce la ubicación de la Dra. López, la Dra. García se dirige hacia ésta. Ya junto a la Dra. López, la Dra. García le pide analizar el expediente del paciente del cuarto 324. La Dra. López acepta realizar la evaluación, por lo que la Dra. García decide abrir el expediente en la pantalla pública que se encuentra ahí. Para esto, es necesario que ambas Dras. sean autentificadas dentro del sistema para poder observar el expediente del paciente en la pantalla pública. Una vez autentificadas, las Dras. evalúan el expediente del paciente. Al finalizar la evaluación, la Dra. López considera el caso de sumo interés, motivo por el cual decide guardar una copia del caso para referencias posteriores.

En base a la información que provee este escenario, se pueden identificar tres componentes: localización, autentificación de usuarios y migración de información.

Escenario 3. Se busca evaluar a distancia el expediente del paciente

El cirujano que atiende al paciente del cuarto 225 sale de imprevisto fuera de la ciudad. La enfermera Martínez, quien atiende al paciente, necesita dar a conocer cierta información al

médico para evaluar el caso clínico del paciente. Para lograr esto, la enfermera decide establecer una sesión de tele-conferencia con el médico para evaluar la situación del paciente. El médico y la enfermera interactúan para evaluar la condición del paciente. Al revisar la información del caso, la enfermera encuentra ciertos datos que pueden servirle como referencia para casos posteriores. Por esta razón, decide almacenar una copia de la información para revisarla con más detalle en otra ocasión.

A partir de la información obtenida en este escenario se pueden identificar dos componentes: colaboración y migración de información.

Escenario 4. Se busca revisar las tareas asignadas al usuario

La Dra. López desea conocer las actividades que tiene asignadas. Para esto, se encuentra dando una ronda por el pasillo donde se encuentra instalada una pantalla pública. Mediante la pantalla pública desea consultar la información referente a sus actividades. Para conocer sus actividades es necesaria su autentificación. Una vez que ha sido identificada, se muestran en la pantalla pública las actividades correspondientes a la Dra. López.

Con la información que provee este escenario se pueden identificar dos componentes: autentificación de usuarios y agenda.

Escenario 5. Un escenario completo

El Dr. Pérez se encuentra trabajando en su oficina, cuando de repente le aparece un mensaje donde se le recuerda la reunión pendiente que tiene con el Dr. González para evaluar las radiografías del paciente del cuarto 226. A la hora indicada, el Dr. Pérez decide establecer una conferencia con el Dr. González, quien se encuentra dando su ronda por el hospital. El Dr. González visualiza en su PDA la imagen del Dr. Pérez, quien le pide revise unas radiografías. El Dr. González abre el archivo de las radiografías para evaluarlas. Dado que la resolución del PDA no es eficiente para evaluar el expediente, el Dr. González decide buscar una pantalla pública para abrir el caso. El Dr. González busca en el mapa la localización de la pantalla pública más cercana. Una vez que localiza la pantalla, el Dr. González migra la información del PDA a la pantalla pública. Ya con la información en la

pantalla, el Dr. González continúa con la evaluación de las radiografías con el Dr. Pérez. Concluida la sesión, el Dr. González decide guardar la información del caso.

De este escenario se pueden identificar cinco componentes: colaboración, migración de información, autentificación de usuarios, localización de servicios y agenda.

III.3 Identificación de componentes

La idea principal en el desarrollo de componentes de software es definirlos de manera lo suficientemente general para que estos puedan ser utilizados en varios contextos. Además, deben ser lo suficientemente simples para que los programadores puedan entender y utilizar las interfaces que estos proveen [Reid et al., 2000].

A partir de esta idea, se analizó el escenario planteado en el punto anterior, y se identificaron cinco componentes principales: localización, autentificación de usuarios, agenda, migración de información, colaboración y control remoto de dispositivos.

En la Tabla III, se describen ciertas características de los componentes tales como: objetivo de los mismos, dispositivos involucrados y funcionalidad base.

Una vez identificados los distintos componentes de software, es necesario revisar la literatura asociada. Esta revisión permite analizar el funcionamiento de otros sistemas o aplicaciones que puedan servir como punto de referencia para el desarrollo de los componentes. En la siguiente sección, se aborda el trabajo relacionado con los componentes de software definidos anteriormente.

Tabla III. Componentes y sus características.

Componente	Objetivo	Funcionalidad
Localización	Localizar la ubicación de usuarios y/o servicios.	Permitir la localización de servicios u otros usuarios de forma gráfica. Mediante un mapa, el usuario podrá visualizar en el dispositivo de su preferencia la ubicación de otros usuarios o servicios que se encuentren dentro de su ambiente.
Autentificación	Validar el acceso del usuario a información y servicios.	Proveer los mecanismos necesarios para validar el acceso del usuario, tanto a su información como a los diversos servicios a los cuales tiene derecho.
Migración	Transferir información entre diversos dispositivos.	Permitir migrar información entre dispositivos heterogéneos, sin que se requiera que el usuario configure los dispositivos.
Agenda	Visualizar las tareas y/o actividades del usuario en diversos dispositivos, así como la calendarización de las mismas.	Este componente pretende tomar las actividades marcadas en la agenda del usuario, para posteriormente desplegarlas de forma gráfica en el calendario que se encuentre en el dispositivo que prefiera el usuario. Así mismo, se busca recordar al usuario las actividades cercanas a su ejecución.
Colaboración	Facilitar la colaboración entre los usuarios.	Proveer de un esquema que permita la fácil realización de videoconferencias, así como trabajo de forma colaborativa en documentos.
Control remoto de dispositivos	Controlar un dispositivo utilizando una PDA.	Este componente permite controlar remotamente el contenido desplegado en una pantalla pública a través de una PDA.

III.3.1 Localización

El componente de localización consiste en proporcionar una aplicación que sirva al usuario para localizar servicios y otros usuarios de forma gráfica, empleando para esto un mapa que muestre la ubicación de dichos elementos. Este componente deberá permitir al usuario desplegar el mapa en diversos dispositivos. Para esto, se tiene pensado que el componente incluya dos implementaciones: una para laptop, pc y pantallas públicas; y otra para PDA. La razón por la cual se piensa en dos implementaciones se debe a las características propias de la PDA, la cual posee una menor resolución que los otros dispositivos.

[Castro, 2005] presenta un sistema de localización el cual permite estimar la posición de un dispositivo móvil a través de una red neuronal que recibe como entrada la intensidad de la señal de seis puntos de acceso inalámbricos. En base a esta información, la red neuronal realiza una transformación no lineal de la información y estima la localización del dispositivo móvil.

El trabajo realizado por [Spreitzer y Theimer, 1993] presenta un esquema de localización de usuarios en un ambiente de cómputo ubicuo. El proyecto consiste en ubicar a los usuarios dentro de un área específica a un alto nivel de granularidad (a nivel cuarto).

La implementación de este proyecto se compone de dos partes: 1) el agente del usuario y 2) el servicio de consulta de la localización. El agente del usuario es el encargado de manejar y controlar la información personal de cada usuario (incluyendo la localización). Permitiendo de esta forma, que las aplicaciones sólo accedan a la información del usuario mediante el agente. El agente del usuario recopila la información utilizando diversas fuentes: sensores, cámaras, información de GPS, dispositivos de entrada, rayos infrarrojos, entre otros. La información que se colecta mediante estos dispositivos sirve para estimar la localización del usuario.

El servicio de consulta de la localización proporciona una forma para ejecutar las consultas que ofrece distintos beneficios relacionados con la eficiencia y privacidad. De esta forma el agente del usuario es quien posee el control de quienes pueden conocer su localización.

[Long et al., 1996] presenta el proyecto Cyberguide el cual consiste en una herramienta que proporciona información a turistas en base al conocimiento de su posición y orientación. El objetivo de Cyberguide es la construcción de aplicaciones móviles que utilicen el contexto del usuario para proporcionarle información relevante.

Esta herramienta consta de dos partes: una interna y otra externa. En el Cyberguide interno se guía al turista en su visita dentro del edificio. El Cyberguide externo guía al turista hacia al edificio. Para lograr su objetivo, el usuario debe poseer un dispositivo GPS en su PDA, para que pueda darse seguimiento a su localización y, de esta forma, mostrarle el camino al edificio.

Cyberguide fue desarrollado en dos versiones: una para Visual Basic y otra para Delphi. Esto debido a que se busca cierta independencia de la plataforma. En cuanto a la información relacionada en la forma en que se implementó la herramienta, se proporcionan datos técnicos de los dispositivos utilizados para estimar la localización del usuario.

III.3.2 Autentificación

El componente de autentificación consiste en un mecanismo que permite validar el acceso del usuario, tanto a información como a servicios que se encuentran inmersos en el ambiente que les rodea.

En el trabajo presentado por [Bardram, 2003] se propone un esquema de autentificación del usuario consciente del contexto, el cual consiste en la autentificación del usuario basado en la proximidad. Este enfoque permite al usuario ser autentificado en un dispositivo simplemente por su proximidad física. Para lograrlo, se proponen tres fases: 1) el uso de JavaCard para identificación y encriptación, 2) la utilización de un sistema consciente del contexto para verificar la localización del usuario, y 3) la implementación de una estrategia de seguridad.

La utilización de JavaCard puede verse como el uso de un token físico empleado para la identificación y la encriptación base para la autentificación. El sistema de consciencia del contexto permite tanto verificar la localización del usuario como desconectarlo del sistema

cuando este deje la sesión abierta y se dirija hacia otro lugar. Los mecanismos de seguridad o recuperación son utilizados para el cambio de fase en el caso de que alguna de las dos fases anteriores fallen, permitiendo de esta forma el acceso del usuario.

La implementación del protocolo de autentificación se ejecuta en una JavaCard, donde cada cliente está equipado con un lector de tarjetas y el protocolo es ejecutado cada vez que el usuario inserta su tarjeta en el lector del cliente. Para lograr la autentificación del usuario, en la tarjeta se almacenan:

- Un identificador del usuario al cual pertenece la tarjeta.
- La contraseña del usuario.
- Una llave secreta y otra pública del usuario.

El sistema consciente del contexto es el encargado de almacenar la información de los usuarios y los derechos a servicios que posee dentro del sistema, así como a los lugares a los cuales tiene acceso. Este esquema permite verificar si el usuario tiene acceso a la información tomando como base el contexto que lo rodea. Esto permite identificar la localización del usuario y verificar si realmente posee acceso a la información y/o servicios que allí se encuentran.

Para garantizar la seguridad del mecanismo de autentificación, se analizaron dos clases de ataques: pasivo y activos. En un *ataque pasivo* se asume que el intruso puede monitorear toda la información entre la tarjeta y la terminal. De esta forma, los datos que el intruso adquiere sirven para apropiarse como usuario legítimo puesto que posee la información de la tarjeta o localización del usuario. En un *ataque activo* el intruso puede borrar, cambiar, agregar o reemplazar completamente tanto la información de la localización o de la tarjeta. Además, el intruso puede crear su propia tarjeta utilizando la información obtenida.

La utilización de estas fases permite la autentificación del usuario para el rápido acceso a la información, sin la necesidad de que este tenga que solicitar explícitamente el acceso a la misma. Puesto que los usuarios aparecen y desaparecen en el sistema, con este esquema se

suprime la solicitud explícita de la información del usuario puesto que toma los datos del contexto que lo rodea para autentificarlo.

La implementación de este proyecto consta de cinco partes: un instalador, un applet de autentificación, un prototipo de pluma personal, el servidor consciente del contexto, y un cliente para el protocolo de autentificación.

El instalador coloca el applet en la tarjeta para luego recuperar la llave pública y almacenarla en el servidor consciente del contexto. El applet de autentificación consiste en habilitar las conexiones al lector de la tarjeta. La pluma personal se utiliza para manejar la JavaCard de forma transparente al usuario.

Tomar como base la implementación de este trabajo para la elaboración del componente de autentificación requiere de infraestructura adicional: plumas personales (personal pen), JavaCard, y lectores de tarjetas. Sin embargo, podrían analizarse nuevas alternativas para suprimir la utilización de estos elementos. Dado que, en el análisis realizado en este trabajo se presentan características interesantes y el enfoque de la validación del usuario basado en la proximidad es factible de implementar, estos podrían tomarse en cuenta para el desarrollo del componente de autentificación.

Sin embargo, existen otros métodos que pueden ser utilizados para autentificar al usuario, tales como: reconocimiento de rostro, reconocimiento de huellas, entre otros.

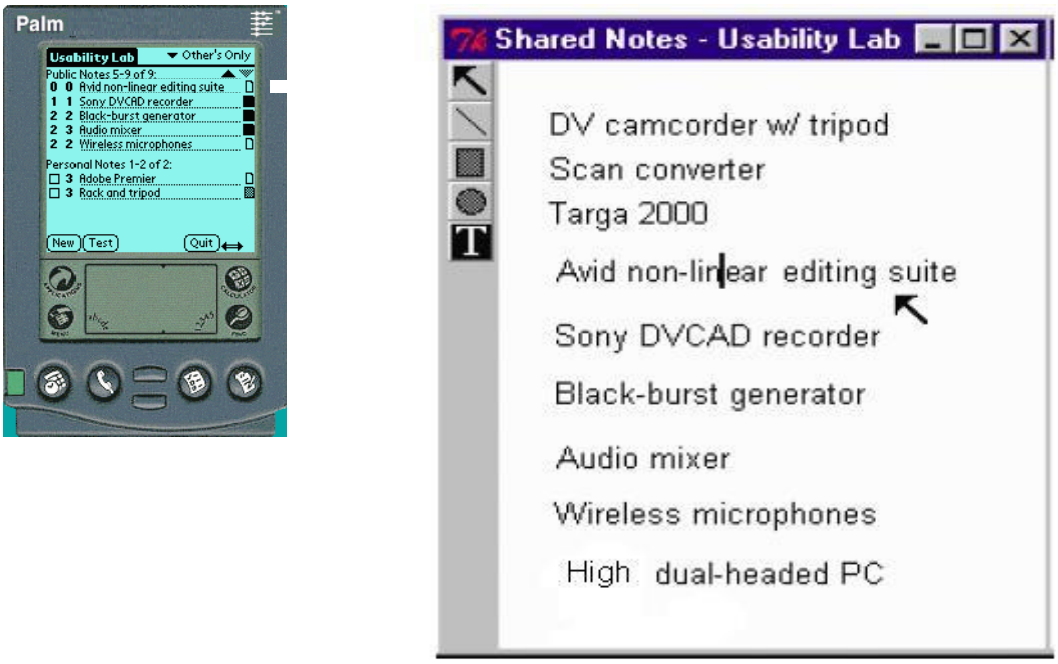
III.3.3 Migración

El propósito del componente de migración es permitir la transferencia de información entre diversos dispositivos que pueden ser heterogéneos.

Existen varios proyectos que utilizan este enfoque. [Greenberg et al, 1999] presenta SharedNotes el cual permite a los usuarios crear y manipular notas personales y públicas entre diversos dispositivos: PDAs, PCs y pantallas públicas. Además, SharedNotes permite la realización de reuniones distribuidas en tiempo real donde los participantes pueden conectarse a la sesión empleando sus estaciones de trabajo. En esta herramienta los

usuarios pueden interactuar unos con otros para agregar notas a la información que se encuentra compartida, permitiendo de esta forma la colaboración síncrona entre los usuarios. Un usuario que utiliza esta herramienta puede transferir la información que posee en su PDA a una pantalla pública por medio de la conexión física de la PDA a la PC. Una vez que la información ha sido transferida a la PC, esta puede ser proyectada a la pantalla pública para que los demás usuarios puedan interactuar con la misma.

Para ejemplificar el uso de SharedNotes obsérvese la figura 8. En la parte izquierda de la figura (8a) se muestra como una persona observa su información personal y pública en su PDA, mientras que en la parte derecha de la figura (8b) se muestra como la misma información es reflejada en una pantalla pública.



8a) Aplicación de SharedNotes para la PDA. 8b) SharedNotes en la pantalla pública.

Figura 8. Implementación de SharedNotes

La publicación de las notas en la pantalla pública se logra mediante la selección de la nota del listado que aparece en el dispositivo del usuario (PDA o PC), de esta forma las personas pueden trabajar en su PDA o PC para trabajos personales y posteriormente parte o todo su trabajo pueda ser visualizado en una reunión a través de una pantalla pública.

Una de las principales desventajas de esta herramienta es que requiere la conexión física entre los dispositivos, lo cual no podría aplicarse a un ambiente de cómputo ubicuo. Además, en este trabajo no se abordan cuestiones técnicas de implementación, simplemente hace mención al uso del GroupKit desarrollado por el mismo Greenberg [Roseman y Greenberg, 1997].

Otro proyecto relacionado a la migración de información es el desarrollado por [Johanson et al, 2001]. En este proyecto se presenta la utilización de multibrowsing, el cual hace referencia a la migración de información web a pantallas públicas. La aplicación desarrollada para el manejo de la información es MB2Go. Esta aplicación permite agregar la funcionalidad de redirección a las opciones que posee el botón derecho del Mouse (ver figura 9). Esta aplicación trabaja para abrir la información fuente obtenida de PCs, PDAs y Laptops al dispositivo origen, o sea, una pantalla pública.

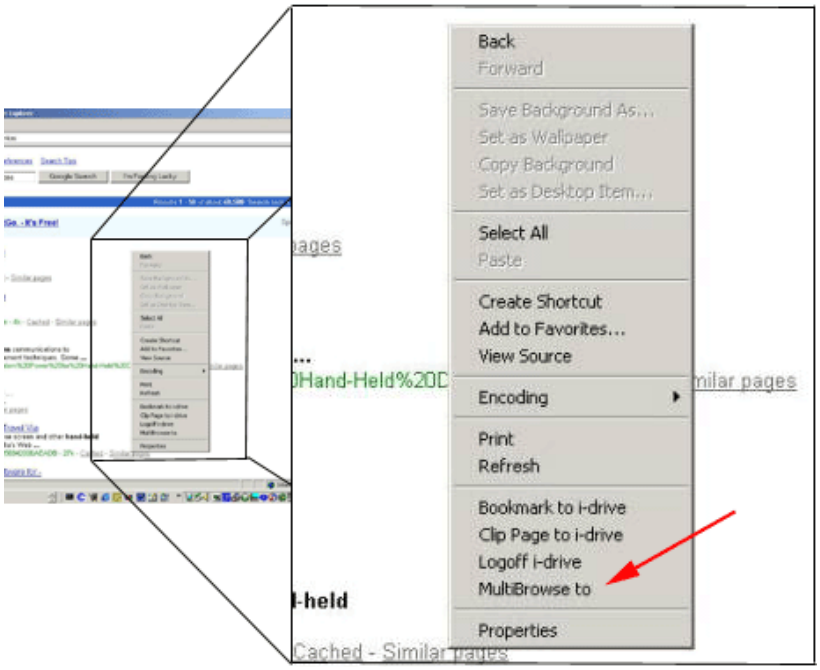


Figura 9. Funcionamiento de MB2Go.

La arquitectura que presenta multibrowsing se compone de un framework, un butler service, el plug-in MB2Go y páginas web para ser multidesplegadas. En el framework de multibrowsing las pantallas públicas pueden asumir uno o más de los tres tipos de roles que

se especifican: 1) cliente regular, 2) cliente aumentado, y 3) destino. El *cliente regular* puede direccionar ligas especiales desde páginas web de propósito diseñado para el despliegue en pantalla públicas. El *cliente aumentado* puede direccionar cualquier página web regular a cualquier pantalla pública disponible en el ambiente. Similarmente, un cliente aumentado puede redireccionar una página web de una pantalla pública hacia si mismo. El *destino* permite a las páginas web ser direccionadas desde un cliente. También permite que páginas web que fueron desplegadas sean eliminadas por clientes aumentados. Los clientes regulares requieren solamente un navegador web para ser ejecutados. Mientras, los clientes aumentados requieren la instalación del plug-in MB2Go en el navegador. Los destinos necesitan ejecutar un servicio especial denominado butler service.

El butler service proporciona diversos métodos para invocar el envío apropiado de eventos, tales como: browse (despliega el URL en un navegador) y getDisplayedResource (regresa el URL que esta siendo desplegado). Estos métodos fueron implementados utilizando APIs COM/OLE exportados por el Internet Explorer. Cada butler service posee un nombre único el cual permanece estático ante reinicios y fallas del sistema.

MB2Go es un plug-in del navegador el cual consiste en una aplicación Java que detecta dinámicamente la disponibilidad de servicios y permite la redirección de las páginas web. MB2Go modifica los registros de Windows para agregar partes adicionales al menú del Internet Explorer. De esta forma, se agrega la opción de “multibrowse link” al menú que posee el botón derecho del mouse. Si el usuario selecciona esta opción aparece una nueva ventana que le permite elegir el destino hacia el cual desea direccionar la liga (ver figura 9). Cuando el usuario selecciona un destino en particular, se envía un evento que causa que el butler service en el destino despliegue la página web solicitada.

Las páginas web pueden ser diseñadas para el propósito específico de ser vistas en pantalla públicas utilizando *multibrowse fat-links,* los cuales son ligas que contienen una solicitud de despliegue de páginas web. Un ejemplo de multibrowse fat-links es el siguiente:

http://gateway/multibrowse?SvcName=Buttler1&OpName=browse&pstsm1=www.stanford.edu

Dando clic a esta liga, se envía una solicitud que requiere que la página http://www.stanford.edu sea desplegada en la pantalla pública que contenga el servicio Buttler1.

Otro trabajo relacionado con esta área es el proyecto realizado por [Baudisch et al, 2003], el cual utiliza las técnicas de drag-and-pop y drag-and-pick para el manejo de pantallas públicas sensibles al tacto o a plumas. El uso de estas técnicas facilita al usuario el manejo de la información de la pantalla, puesto que le permite seleccionar la información y arrastrarla a un nuevo destino.

En drag-and-pop, la técnica consiste en dar una respuesta temporal a la solicitud de arrastre de un icono a otro icono del usuario. Esto es, se mueven temporalmente los iconos destino potenciales hacia la localización actual del cursor del usuario, de tal modo que permite al usuario interactuar con estos iconos usando pequeños movimientos. Como ejemplo (ver figura 10), tómese el caso en que el usuario desea borrar un documento de Word simplemente arrastrándolo hacia la papelera de reciclaje (10a). Cuando el usuario inicia el arrastre del icono hacia la papelera de reciclaje, los iconos que son compatibles a este tipo y que se localizan en la dirección del arrastre del usuario son resaltados y movidos hacia la posición del cursor (10b). El usuario arrastra el documento hacia la papelera y libera el botón del mouse (10c). La papelera acepta el documento. Cuando el usuario elimina el icono todos los demás iconos que fueron marcados desaparecen de forma instantánea (10d).

Drag-and-pick permite a iconos activos abrir carpetas o aplicaciones. Esta técnica inicia cuando el usuario realiza una interacción de arrastre hacia un espacio vacío en la pantalla. La respuesta del sistema a esta interacción de arrastre es similar a la de drag-and-pop, sin embargo, posee ciertas diferencias. Primero, todos los iconos localizados en la dirección del movimiento de arrastre serán resaltados, no sólo aquellos que son compatibles (ver figura 11). Segundo, cuando el usuario arrastra el cursor del mouse sobre uno de los destinos y libera el botón, la carpeta, archivo o aplicación asociada con el icono es activada como si el usuario hubiera dado doble clic sobre ésta.

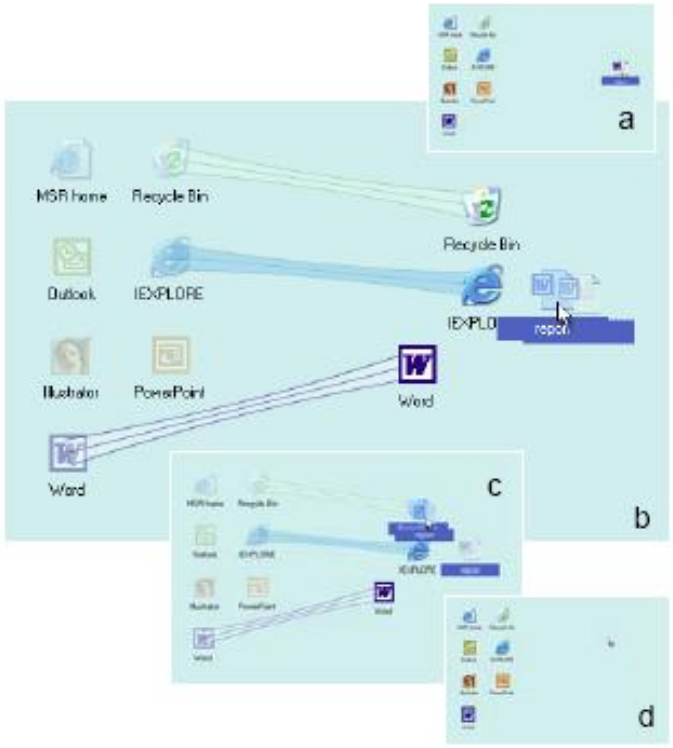


Figura 10. Uso del Drag-and-pop.

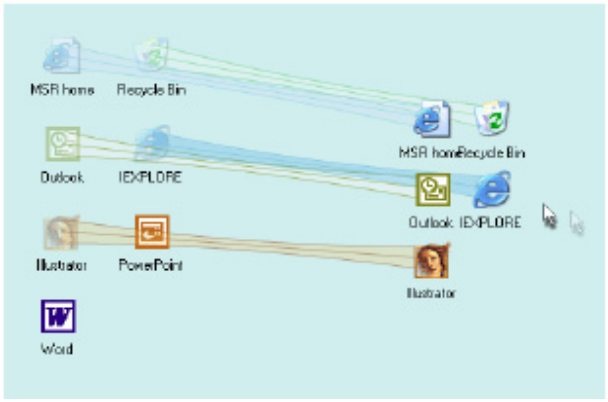


Figura 11. Uso del Drag-and-pick.

La implementación de drag-and-pick se pensó en un principio que fuera aplicable para cualquier tipo de widget, ejemplos: cualquier botón y menús localizados en un monitor que no soporta las interacciones de plumas. Para ilustrarlo, en la figura 12 se muestra como esto permitiría al usuario el uso de la pluma para abrir una aplicación, donde el icono origen que

desea esta localizado en un monitor distinto al destino que desea. Sin embargo, esta funcionalidad aún no ha sido implementada.

Para transportar información, esta puede ser conectada a un passenger para ser trasladada físicamente a una nueva localización. Después, simplemente poniendo el passenger en un dispositivo especial llamado *bridge*, la información conectada es traída inmediatamente desde una base de datos y desplegada en la pantalla que corresponde al bridge.

Figura 12. Drag-and-pick utilizando diversos dispositivos.

En cuanto a la implementación, no se tienen detalles técnicos ni información relacionada a como fue desarrollada esta herramienta.

[Konomi et al, 1999] introduce el mecanismo de passage. Passage proporciona una forma fácil e intuitiva para transportar varios tipos de objetos digitales utilizando objetos físicos normales sin la necesidad de etiquetas especiales de identificación. Estos objetos físicos, denominados *passenger*, pueden ser vistos como un bookmark físico en un mundo virtual.

Cualquier objeto físico puede ser un passenger, la única restricción es que estos deben poder ser detectados por un bridge e identificados de forma única por el sistema. Como ejemplo, en la figura 13 se puede observar a una persona sentada frente a su PC utilizando un llavero como passenger, el cual es depositado en el bridge correspondiente. El bridge consta de una parte física donde se coloca el passenger, y una parte virtual donde aparece una ventana en la pantalla. Para ilustrar la recuperación y despliegue de información puede

observarse la figura 14. En esta figura la recuperación y despliegue de la información se lleva a cabo después de que el passenger ha sido colocado en el bridge.

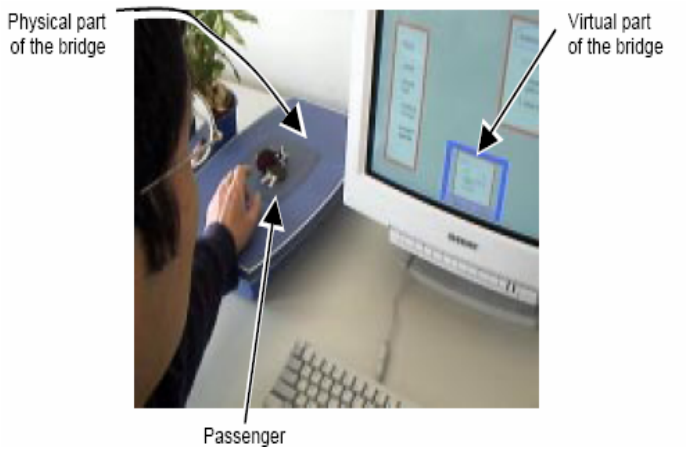


Figura 13. Ilustración del uso de passenger y bridge

Figura 14. Recuperando y desplegando información

Para detectar e identificar los objetos físicos o passengers, se emplean dos métodos: por peso o por etiquetas electrónicas de identificación. La identificación de los objetos por peso consiste en que los bridges están equipados con básculas electrónicas que permiten medir el peso exacto de los objetos. La identificación de objetos por etiquetas electrónicas de identificación consiste en equipar a los bridges con dispositivos de identificación libres de contacto (MIKRON's EasyKey) para que estos puedan leer las etiquetas electrónicas de identificación.

La conexión de objetos físicos e información digital se realiza mediante agentes passage los cuales utilizan los datos recibidos desde un dispositivo y la base de datos. Estos agentes son los encargados de obtener las propiedades físicas de un passenger, tales como el peso y/o el identificador para permitir la recuperación de la información.

La implementación de este mecanismo requiere de agentes passage, una base de datos y la configuración de los agentes. Para esto, los autores incluyen información técnica referente a la implementación de los mismos.

En la Tabla IV se muestra un resumen de los sistemas descritos anteriormente relacionados con el componente de migración.

Tabla IV. Resumen de sistemas relacionados con la migración de información.

Sistema /aplicación	Características	Dispositivos que aplica	Funcionamiento	Técnica que aplica	Limitantes
SharedNotes [Greenberg et al., 1999, University of Calgary]	Permite la creación y manipulación de notas personales y públicas.	PDAs PCs Pantallas públicas.	Un usuario puede transferir información de su PDA a una pantalla pública. Una vez transferida la información, esta es proyectada en el pantalla donde los demás usuarios pueden interactuar con la misma y agregar notas.	Selección de la información por medio de checkbox.	▪ Requiere conectividad física entre dispositivos. ▪ No hace referencia a la implementación.
MB2Go [Johanson et al., 2001, Stanford University]	Permite la migración de información web a pantallas públicas.	PCs PDAs Laptops Pantallas públicas.	Para migrar la información de una página web a una pantalla pública, el usuario debe presionar el botón derecho del mouse para que se muestren las opciones. Una vez que aparecen las opciones el usuario debe seleccionar "Multibrowsing" y elegir el destino hacia el cual desea proyectar la página web.	Opción de despliegue agregada en el menú del botón derecho del mouse.	▪ Aplica sólo para páginas web. ▪ Requiere acceder a los registros del sistema operativo.
Drag-and-pop and drag-and-pick [Baudisch et al., Microsoft]	Permite la manipulación de iconos para facilitar el manejo de la información en pantalla.	Pantallas públicas.	El usuario selecciona la información que se encuentra en la pantalla (iconos) y los arrastra hacia un nuevo destino.	Drag-and-pop y drag-and-pick.	▪ Se encuentra en fase de elaboración ▪ No hace referencia a la implementación.
Passage [Konomi et al., GMD - IPSI]	Proporciona un medio para el transporte de información entre objetos digitales utilizando objetos físicos (passengers).	PCs Pantallas públicas	La información es conectada a un passenger para su traslado físico a una nueva localización. Después, el usuario debe poner el passenger en un dispositivo especial llamado *bridge*, que permite *traer* la información y desplegarla en la pantalla que corresponde al bridge.	Objetos que sirven como identificadores para la transferencia de información	▪ Requiere de un objeto físico y un lector para identificar el objeto

III.3.4 Agenda

El componente de agenda tiene como objetivo la calendarización de las tareas y/o actividades del usuario independientemente del dispositivo que este utilice para agregarla en su lista de actividades. Además, en este componente se pretende que el usuario pueda visualizar la información de sus actividades o tareas en el dispositivo de su elección (PDA, laptop, PC, pantalla pública).

El trabajo realizado por [Dey y Abowd, 2000], CybreMinder, es una herramienta consciente del contexto que da soporte al usuario en el envío y recepción de recordatorios que pueden ser asociados a situaciones que involucran cierto contexto tal como el tiempo, lugar y algunos otros.

CybreMinder consta de dos partes: 1) creación del recordatorio y 2) entrega del recordatorio. En la creación del recordatorio, el usuario cuenta con una interfaz gráfica que permite la captura de la información relacionada con el recordatorio, dicha interfaz se asemeja al esquema que presenta un mensaje de correo electrónico. En la figura 15 puede observarse la interfaz que se presenta al usuario para la captura del recordatorio. En esta parte, el usuario puede especificar el título, hacia quien(es) va dirigido el recordatorio, la prioridad, el mensaje y la fecha de expiración del mensaje, la cual indica la fecha y hora en que expira el recordatorio, y es cuando este debe ser entregado en caso de que no haya sido enviado anteriormente.

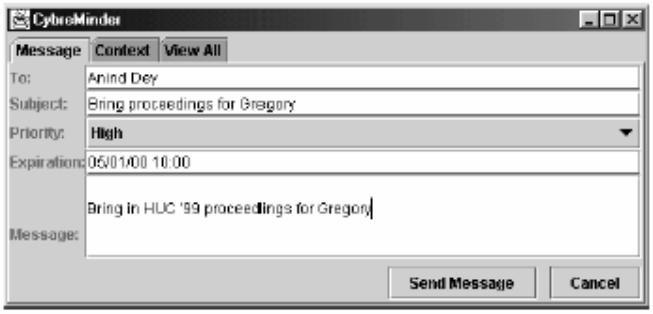


Figura 15. Interfaz para la captura del recordatorio en CybreMinder.

Una vez que el usuario ha capturado la información que va a ser enviada en el recordatorio, es posible definir el contexto en el cual este va a ser entregado. En la figura 16 se muestra

la interfaz que permite la construcción dinámica de una situación arbitraria o de un contexto asociado con el recordatorio a ser entregado. Para esto, la infraestructura de CybreMinder da soporte a una serie de situaciones, las cuales se toman como base para la especificación del recordatorio.

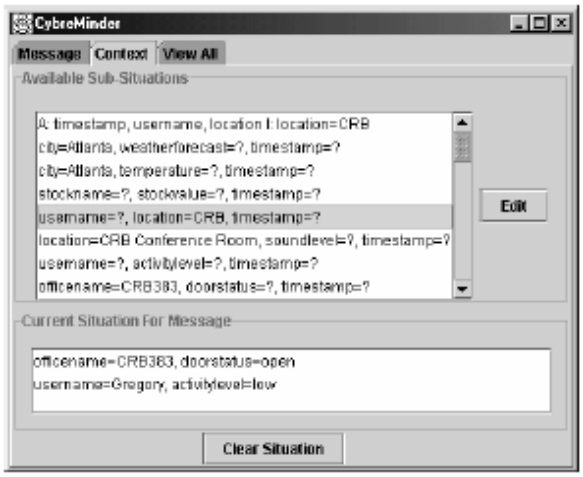


Figura 16. Interfaz para especificar el contexto o la situación del recordatorio

La entrega del recordatorio se realiza cuando se satisface con la situación establecida o cuando expira el recordatorio. Algunas condiciones bajo las cuales la entrega del recordatorio puede ser especificada son: fecha, hora, y localización en la cual se encuentra el usuario que recibirá el mensaje, entre otras. Es posible cambiar el estado del recordatorio. Para esto, se provee al usuario de una interfaz (ver figura 17) donde este puede modificar el estado del recordatorio.

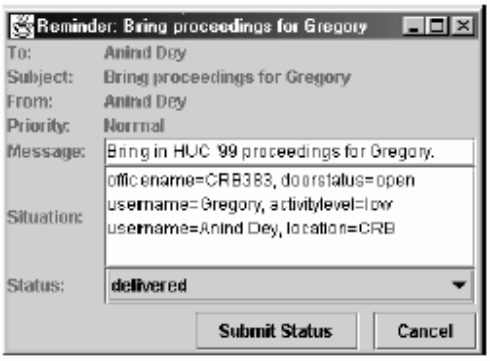


Figura 17. Interfaz para modificar el estado de un recordatorio

En cuanto a la implementación, esta herramienta fue desarrollada utilizando la arquitectura que provee el Context Toolkit bajo el lenguaje de programación Java. Context Toolkit es un toolkit que da soporte a la construcción de aplicaciones conscientes del contexto. La utilización de Java da soporte a la entrega de recordatorio en múltiples dispositivos tales como: PCs, teléfonos celulares, PDAs, e impresoras. Sin embargo, se piensa como trabajo futuro desarrollar implementaciones de CybreMinder para Palms, beepers, teléfonos celulares, los cuales puedan capturar la información mediante el teclado, pluma o voz. Además, se busca permitir la creación automática de recordatorios a partir de la lista de actividades del usuario y el calendario.

Otro trabajo relacionado es el de [Bellotti et al., 2004] en el cual presentan la herramienta TaskVista. TaskVista permite la colección y el listado de las actividades del usuario, así como de tareas asignadas a estas actividades.

El funcionamiento de la herramienta se basa en la captura de actividades mediante el teclado o arrastrando un elemento (archivo o correo electrónico) a la lista. Este último método establece el título o encabezado de la actividad por defecto. La prioridad de las actividades es establecida en base al orden de despliegue. Sin embargo, dicho orden puede ser cambiado por el usuario.

Una característica importante que presenta esta herramienta es la extracción de los nombres de los participantes de las actividades, los cuales son extraídos de la información del recipiente de correos electrónicos o del contenido de documentos. De esta forma, se permite establecer una lista de contactos para cada tarea, permitiéndole fácilmente al usuario crear un mensaje para todos los participantes. En la figura 18, se puede observar la interfaz de TaskVista y los componentes que esta posee.

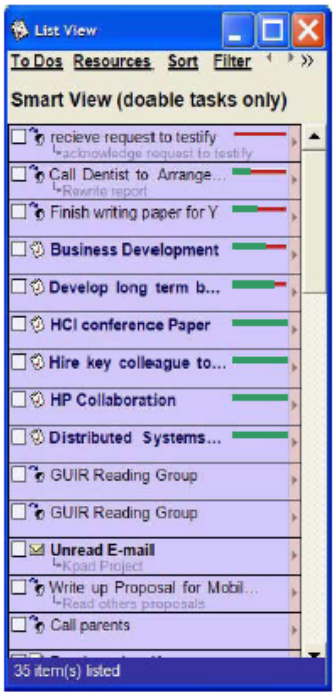


Figura 18. Interfaz de TaskVista.

En cuanto a la implementación y detalles técnicos de la herramienta, solamente se menciona que TaskVista fue implementada en C#.

[Landry et al., 2004] presentan la herramienta TaskMinder. TaskMinder es un sistema administrador de tareas que utiliza información contextual acerca del usuario para proporcionar mejor soporte a la administración de tareas mediante la prioritización automática de las mismas. Esta herramienta utiliza el contexto del usuario para auxiliarlo en la determinación de que tarea puede ser direccionada dado su contexto actual. De esta forma, el usuario puede proveer retroalimentación al sistema acerca de una tarea en particular mediante un conjunto de botones que aparecen con cada tarea (ver figura 19).

El funcionamiento de TaskMinder se basa en la captura de la tarea a través de la pantalla "*Task Entry"* (ver figura 20). Dicha pantalla proporciona un campo para que el usuario introduzca la descripción de la tarea, así como un calendario para que se especifique la fecha de realización. De acuerdo a esta fecha, el usuario puede clasificar la tarea de acuerdo a la importancia y urgencia de la misma.

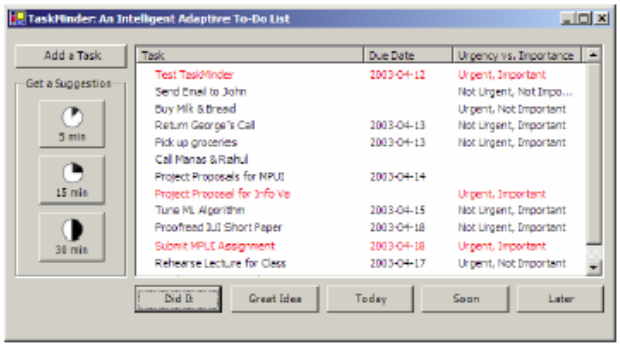


Figura 19. Lista de actividades capturadas en TaskMinder.

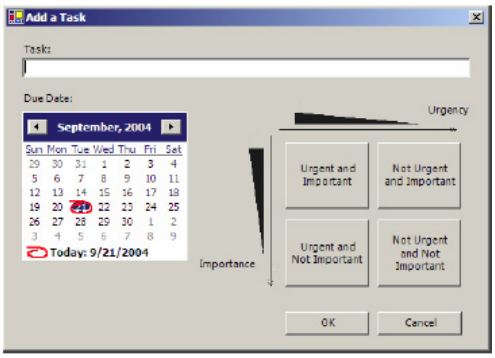


Figura 20. Interfaz de Task Entry, que permite la captura de las tareas.

Una vez que las actividades han sido capturadas, el usuario puede visualizar la lista de las tareas que introdujo en el sistema (ver figura 19). Dicha lista es desplegada en orden descendiente de importancia. En esta lista se proporciona el soporte para la consulta de tareas basadas en el tiempo que estas tienen disponibles.

En lo referente a la implementación, sólo se menciona que TaskMinder utiliza el modelo cliente-servidor. Donde, el cliente obtiene la información del contexto referente a la actividad que realiza, el tiempo disponible, la localización y el tiempo actual. El servidor es el encargado de almacenar la información contextual y las peticiones del usuario. Sin embargo, no se describe como se realizó la implementación ni bajo que lenguaje fue desarrollada.

En la Tabla V se muestra una breve síntesis de los sistemas relacionados con el componente de la agenda.

Tabla V. Resumen de sistemas relacionados con agendas.

Sistema /aplicación	Características	Dispositivos que aplica	Funcionamiento	Técnica que aplica	Limitantes
CybreMinder	Permite el envío y recepción de recordatorios conscientes del contexto.	PCs, Celulares, PDAs, Impresoras	El usuario debe crear el recordatorio capturando: destinatario, prioridad, título, mensaje, fecha de expiración, entre otros. Una vez que se captura esta información se envía el mensaje, al cual se le asocia el contexto especificado anteriormente. El mensaje se entrega de acuerdo al contexto que fue especificado.	Asignación de prioridad, fecha de expiración y consciencia del contexto.	▪ Afecta la disponibilidad del usuario
TaskVista	Permite la colección y el listado de las actividades del usuario. Permite la extracción de nombres de usuario.	PC	El usuario captura el listado de sus actividades, las cuales pueden ser introducidas por teclado o arrastrando el elemento a la lista.	Prioridad de las actividades	▪ No hace referencia a la implementación
TaskMinder	Utiliza información contextual para dar soporte a la administración de tareas.	PC	El usuario captura la información de la tarea a la cual le es asignada cierto contexto, como: localización, relevancia, fecha, etc. Una vez que son especificados estos datos, la tarea se entrega conforme se cumpla con el contexto especificado anteriormente.	Prioridad de tareas	▪ No hace referencia a la implementación

III.3.5 Colaboración

El componente de colaboración consiste en una herramienta de apoyo que facilite la colaboración entre los usuarios. Para lograr esto, dicho componente deberá dar soporte para audio, video, transferencia de datos, compartir aplicaciones, entre otros mecanismos utilizados para que los usuarios puedan colaborar sin preocuparse de cómo éstos se realizarán.

Existen diversos trabajos que abarcan cada uno de los elementos o factores que involucra la colaboración. El iRoom presentado por [Johanson et al., 2002] se enfoca a la interacción entre múltiples pantalla públicas y PCs, laptops, PDAs y otros dispositivos portátiles en un salón de conferencias con ambiente aumentado. El iRoom contiene varias pantallas públicas montadas sobre la pared, una pantalla pública en una mesa, red inalámbrica, mouse y teclados inalámbricos para controlar las pantallas públicas. Además, cuenta con dispositivos especializados de entrada y salida, tales como: micrófonos, cámaras, apuntador láser, entre otros. El objetivo de este proyecto es el habilitar ambientes multi-usuario multi-dispositivos que faciliten la fácil adición de pantallas públicas y dispositivos de entrada, para que de esta forma se permita la migración de cualquier trabajo entre los dispositivos.

La implementación del iRoom recae sobre la infraestructura que provee el iROS (Interactive Room Operating System) el cual consta de tres subsistemas principales: Data Heap, iCarfter, y Event Heap. Dichos subsistemas están diseñados para mover datos y control, y para coordinar dinámicamente las aplicaciones.

Las limitantes que presenta esta herramienta incluyen la falta de mecanismos para el almacenamiento y monitoreo del estado de los servicios, así como la seguridad necesaria para interactuar en ambientes interactivos puesto que, la comunicación indirecta hace los mensajes públicos, lo cual invade la privacidad de la información del usuario.

En el trabajo de [Streitz et al., 1999] se presenta el ambiente i-LAND: un paisaje interactivo para la creatividad e innovación. i-LAND requiere y proporciona nuevas formas de interacción humano-computadora (HCI: Human-Computing Interaction) y nuevas

formas de trabajo cooperativo asistido por computadora (Computer-Supported Cooperative Work). Su diseño está basado en la integración de espacios de información y arquitectónicos, implicaciones de nuevas prácticas de trabajo y un estudio de requerimientos empírico.

i-LAND consiste de varios componentes *roomware*, tales como: una pared electrónica interactiva, una tabla interactiva, dos sillas computarizadas, entre otros. La utilización de estos componentes permite: establecer relaciones entre objetos reales y virtuales en los espacios físicos y en objetos de información digital en el espacio de información virtual; implementar nuevas prácticas de trabajo resultado de innovaciones organizacionales y de los requerimientos para el diseño de espacios de trabajo colaborativos; proveer un framework tecnológico que proporcione los elementos necesarios para desarrollar ambientes de realidad aumentada y de cómputo ubicuo.

El propósito de i-LAND es habilitar un ambiente de cómputo ubicuo para que los usuarios puedan interactuar utilizando diversos dispositivos que facilitan el trabajo que realizan. Las aplicaciones desarrolladas para lograr el funcionamiento de estos dispositivos están basadas en el software BEACH [Tandler, 2003].

Otro trabajo relacionado con este tema es el desarrollado por [Hooper, 2003]. En este trabajo se presenta el componente NDNM3 que incrementa la funcionalidad del NetMeeting. Dicho componente da soporte a: audio, video, datos, transferencia de archivos, compartir aplicaciones, seguridad de datos. Sin embargo, no existe mucha información relacionada con este componente. Por tal motivo, no se conocen mayores características ni la forma en que fue implementado.

III.3.6 Control remoto de dispositivos

El componente de control remoto de dispositivos es una herramienta que permite controlar dispositivos de forma remota utilizando una PDA. Mediante la PDA, se pueden llevar a cabo distintas acciones, cuyos resultados serán reflejados en el dispositivo que está siendo controlado. El funcionamiento del componente es el siguiente: la PDA se conecta al

dispositivo que desea controlar. Una vez que la PDA está conectada, el dispositivo envía la información que está siendo visualizada en su pantalla. De esta forma, la información que se envía a la PDA es la misma que la que aparece en el dispositivo. Por tal razón, el usuario a través de la PDA puede controlar y/o manejar la información del dispositivo. Los cambios o acciones a la información realizados en la PDA son reflejados en el dispositivo que está siendo controlado.

Este componente de control remoto de dispositivos es presentado por [Markarian, 2005]. Sin embargo, existen diversos sistemas y/o aplicaciones que han sido desarrollados para controlar de forma remota dispositivos. [Paradiso y Feldmeier, 2001] presentan un dispositivo eléctrico que permite la trasmisión inalámbrica de un código (ID), el cual es enviado mediante la pulsación de un botón. Este dispositivo, tiene el potencial de habilitar controles e interfaces que son introducidas en ambientes interactivos sin la necesidad de conexiones, baterías o señales ópticas o acústicas.

Otro trabajo relacionado, es el presentado por [Kaowthumrong et al., 2002], el cual se enfoca al modo de interacción que se presenta con el control remoto de dispositivos a través de PDAs o teléfonos celulares. Los autores analizan el problema del dispositivo activo, el cual se refiere a conocer que dispositivo está activo en un ambiente saturado de dispositivos que pueden ser utilizados por el usuario. Para solucionar este problema, utilizan un enfoque de selección automática en base a la información histórica del contexto del dispositivo. Esta información sirve para aplicar algoritmos de predicción y, así de esta forma, poder determinar automáticamente el siguiente dispositivo activo con el cual el usuario intentará interactuar. Los escenarios que se presentan en este trabajo contemplan el control remoto de: televisiones, dvd, alarmas y estereos, mediante el uso de un dispositivo personal (PDA o un teléfono celular) en donde se proporcionan las interfaces necesarias para controlar diversas acciones sobre los distintos dispositivos.

[Villar et al., 2005] presentan "*The Pendle"*, un nuevo modelo de interacción para ambientes aumentados basado en una mezcla de interacción intuitiva y un dispositivo personal, el cual posee la capacidad de reconocer gestos manuales. Mediante el intercambio

del control entre el ambiente y el usuario, el modelo de interacción combina las ventajas de la manipulación directa con el poder de los ambientes pro-activos basados en sensores los cuales excluyen la carencia de control del usuario y personalización. *The Pendle,* actua como mediador entre su ambiente y el usuario, y proporciona una interfaz que facilita la interacción casual. Los autores presentan dos ejemplos, el *MusicPendle* y el *NewsPendle*, los cuales demuestran las ventajas de la personalización de información del usuario mediante el uso del *pendle.* En estos ejemplos, se ilustra como el *pendle,* permite controlar los distintos dispositivos asociados a cada uno de los ejemplos.

III.4 Middleware y componentes

Middleware es el nombre que recibe el conjunto de servicios que se encuentra situado como capa intermedia entre el sistema operativo y la plataforma de programación. En la actualidad, el desarrollo de middlewares aislados que ofrezcan servicios específicos para aplicaciones específicas está desapareciendo. En su lugar, está emergiendo el desarrollo de servidores especializados que combinen las funciones del middleware con frameworks de componentes específicos. Existen tres tipos de servidores: de aplicación, de integración, y de flujo de trabajo y organización [Szyperski et al., 2002].

Los servidores de aplicación combinan: administración de aplicaciones, transacción de datos, balanceo de carga, entre otros. Los servidores de integración combinan: conversión de protocolos, conversión de datos, y ruteo, entre otros. Los servidores de flujo de trabajo y organización combinan: eventos de ruteo, toma de decisiones y otros.

En general, estos servicios o middlewares para componentes deben habilitar patrones de interacción síncronos y asíncronos. Los mecanismos de comunicación que utilizan están basados en eventos, llamadas a procedimientos, o envío de mensajes.

En la siguiente sección, se presentan algunos ejemplos del trabajo relacionado con el desarrollo de componentes basados en middleware.

III.4.1 Trabajo relacionado

J2EE y .NET/COM+ se han enfocado a la implementación de servicios para cumplir con la demanda del mercado de servidores de aplicaciones. Para el mercado del flujo de trabajo y organización del servidor existe una gran variedad de productos incluyendo: IBM's MQ Series Workflow y Microsoft's BizTalkServer. El mercado del servidor de integración es el mas fragmentado, con productos de muchas formas de varias compañías incluyendo: CrossWorlds, IBM (WebSphere B2B Integrator), Microsft (BizTalk Server), Oracle (XML Integration Server), SeeBeyond (eBussines Integration Suite), Sun (Supone Integration Server), Sybase (Integrator), Tibio (ActiveEnterprise), Vitria (BusinessWare), webMethods (Enterprise), y WRQ (Verastream).

III.4.2 Middleware SALSA

SALSA [Rodríguez et al., 2005], es un middleware que facilita la implementación y evolución de aplicaciones bajo un ambiente inteligente. Además, SALSA ofrece mecanismos para el intercambio de información y la comunicación entre agentes.

Una de las principales ventajas de SALSA es la facilidad que proporciona a los programadores para construir agentes autónomos bajo ambientes de cómputo ubicuo, mediante un API (Application Program Interface) que facilita: la composición, envío y recepción de mensajes entre agentes, una infraestructura de software para agentes y mecanismos para interactuar con servicios, usuarios y otros agentes mediante el uso de un protocolo estándar.

La arquitectura de SALSA consiste de varias capas tal como se ilustra en la figura 21. La primera es la plataforma de comunicación, llamada Broker de Agentes que coordina la interacción entre los agentes. El API (Application Programming Interface) de SALSA incluye un conjunto de clases que facilitan la implementación: de los subsistemas del agente (percepción, razonamiento y acción), las interacciones entre agentes y de otras características que un agente residiendo en un ambiente de ubicomp puede poseer, tal como derivar contexto o movilidad. Finalmente, SALSA proporciona un servicio de Directorio

de Agentes, que los agentes utilizan para registrarse o buscar otros agentes a través de sus atributos o por los servicios que ofrecen.

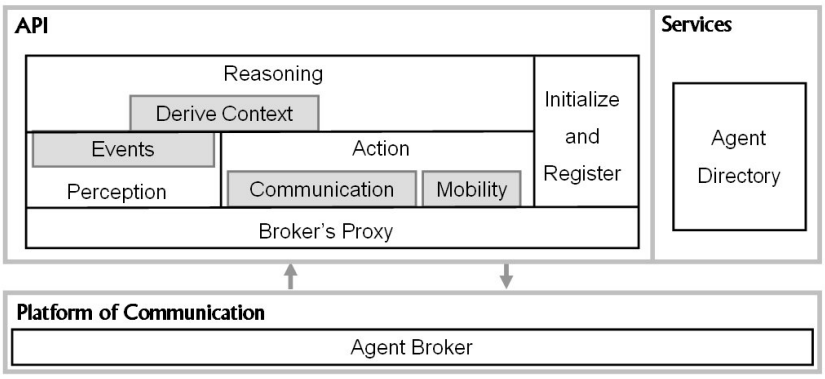


Figura 21. Arquitectura de SALSA.

Sin embargo, la principal desventaja que presenta SALSA es el bajo nivel de abstracción que posee. Esto debido a que el alcance del dominio de la aplicación se limita a la funcionalidad que presenta el conjunto de clases, las cuales para ser implementadas requieren amplio conocimiento por parte del desarrollador. Esta, es una de las principales razones por las cuales se busca desarrollar este trabajo, cuyo propósito es proporcionar un conjunto de componentes que permitan al desarrollador elaborar aplicaciones ubicuas de forma fácil y sencilla, donde éste no requiera conocimiento profundo acerca de cómo implementarlas.

Para iniciar con el desarrollo de componentes, se eligió implementar el componente de migración. Este componente se desarrolló en base a la funcionalidad que ofrece el middleware de SALSA. Dentro de las principales características que ofrece el componente de migración se tiene: flexibilidad, configuración y reconfiguración, encapsulación, manejo de consciencia del contexto, ejecución independiente, múltiples usos, composición con otros componentes.

III.5 Resumen

En este capítulo se presentó la introducción al cómputo ubicuo. Partiendo de esto, se presentaron algunos escenarios ubicomp de donde se identificaron y definieron algunos

componentes, los cuales sirvieron como base para el estudio de sistemas y aplicaciones relacionadas con cada uno de los componentes. Para finalizar, se presenta la relación entre middlewares y componentes, así como el trabajo relacionado en esta área.

Para continuar con este trabajo, se eligió continuar con el desarrollo del componente de migración. En el siguiente capítulo, se describe el análisis de dicho componente.

Capítulo IV

Análisis del Componente de Migración

IV. 1 Introducción

En este capítulo se presenta el análisis del componente de migración. Dicho análisis contempla el estudio de distintas aplicaciones que sirven como ejemplo del uso de dicho componente. Estos ejemplos sirven como base para definir una serie de escenarios de uso. El objetivo de dichos escenarios es describir situaciones en las cuales puede presentarse la migración de información de manera que el diseño del componente sea aplicable a una variedad de situaciones.

A partir de la información obtenida con el estudio de los ejemplos de aplicaciones y de los escenarios de uso, es posible definir el servicio principal que provee el componente de migración: *transferencia de información* entre dos dispositivos.

Los requerimientos que debe cumplir el componente, son establecidos a partir del análisis de la información obtenida en la definición del servicio, los ejemplos de aplicaciones y los escenarios de uso. Dichos requerimientos definen las reglas o normas que debe cumplir el componente.

Un punto muy importante es la definición de las entradas y salidas del componente. Dichas entradas y salidas se establecieron en base a la funcionalidad que debe proveer el componente de migración. Con toda esta información, se definieron los casos de uso del componente y se establecieron las interacciones con otros componentes.

En las siguientes secciones, se describe el desarrollo de cada uno de los puntos citados anteriormente.

IV.2 Ejemplos de aplicaciones

El primer punto dentro del análisis del componente de migración, es identificar ejemplos de aplicaciones donde éste pueda ser utilizado. Por esta razón, se analizaron distintos ejemplos de aplicaciones. Dichos ejemplos, permiten entender mejor el funcionamiento y utilización del componente de migración.

IV.2.1 Intercambio de información entre dispositivos

En un ambiente de cómputo ubicuo existe una gran variedad de dispositivos heterogéneos inmersos en el ambiente con los cuales puede interactuar el usuario. Debido a esta razón, es importante que se faciliten los medios para que el usuario pueda transferir información de un dispositivo a otro. La transferencia de información debe permitir el fácil y rápido intercambio de información entre los dispositivos.

Como ejemplo, supongamos que el usuario desea proyectar una presentación de su laptop a una pantalla pública. La finalidad de proyectar la presentación en la pantalla es permitir a los demás participantes de la reunión visualizar la información. Similarmente, pudiera presentarse el caso en que existe cierta información publicada en la pantalla la cual resulta de interés para el usuario. Y por tal motivo, éste desea almacenar una copia de la información en su laptop.

IV.2.2 Manejo de información pública vs. privada

En los dispositivos de uso personal tales como: PDA, PC, teléfonos celulares, laptops, entre otros, los usuarios tienden a llevar un registro de las tareas y/o actividades a realizar. Mediante la agenda, los usuarios calendarizan las actividades pendientes estableciendo fecha, hora, participantes, lugar, tipo de la reunión, entre otros elementos. Llevar un control de esta manera facilita al usuario la consulta de las actividades pendientes, puesto que

existen diversas aplicaciones que permiten visualizar gráfica y concentradamente las actividades programadas previamente.

En algunas ocasiones el usuario interactúa con dispositivos públicos (tales como pantallas) donde existen aplicaciones que transfieren información del dispositivo personal (en este ejemplo, de la agenda) del usuario al ambiente público. La transferencia de información de un dispositivo personal a un dispositivo público implica ciertos riesgos dado que, la información contenida en el dispositivo personal del usuario no es propiamente del dominio público. Por esta razón, es necesario adaptar la información del usuario para que en los dispositivos públicos sólo pueda visualizarse la información que se considere pública, o bien, efectuar cambios para que la información personal que se despliegue en el dispositivo público no pueda ser entendida por otras personas.

IV.2.3 Publicación de la información

Cuando un usuario publica cierta información pierde el control sobre esta. De manera similar, el control se pierde si se proporciona información a otra persona puesto que, no se lleva el registro o no se da seguimiento sobre a quien podría pasarle la información. Para solucionar este problema, la información podría contener detalles sobre quien tiene derecho a accederla y los privilegios que posee. De esta forma, transferir información de un usuario a otro puede ser mantenido bajo control.

Por ejemplo, supongamos que un usuario desea publicar la información de su artículo en una pantalla pública para que los demás usuarios puedan observar la información y hacer comentarios. Puesto que, el autor del artículo es quien posee el control sobre el documento, esté decide que los demás usuarios solamente tendrán derecho a visualizar el artículo y a hacer comentarios en la pantalla. Restringiendo de esta forma los privilegios de los demás usuarios quienes no podrán: editar la información, copiarla, almacenarla o imprimirla, única y exclusivamente podrán ver el contenido y agregar comentarios. De esta forma, el autor asegura el control sobre la información.

De igual manera, el autor puede elegir a las personas que tendrán acceso al documento y los privilegios que estos tendrán sobre el mismo. Como ejemplo: el autor desea que otro usuario revise la información del artículo, por este motivo le transfiere la información a su laptop solamente con el privilegio de edición, de esta forma se asegura que no pueda copiar, almacenar o imprimir la información.

IV.2.4 Presentación adaptable de la información

En las PDAs, teléfonos celulares, entre otros dispositivos pequeños y móviles, el espacio de visualización está limitado por el tamaño que posee el dispositivo. Además, debido a la capacidad de almacenamiento que presentan dichos dispositivos, la transferencia de información puede verse afectada, dado que, se limita el tamaño del archivo que se desea transferir. Para solucionar este problema, se han adoptado algunas técnicas que permiten adaptar la información dependiendo del tipo de dispositivo a utilizar.

Por ejemplo, supongamos que existe cierta información desplegada en una pantalla pública la cual contiene algunas imágenes e información de nuestro interés. Debido a que la información es útil para nuestro caso, deseamos almacenar una copia de dicha información en nuestro PDA. Sin embargo, dadas las limitantes de almacenamiento y despliegue que posee nuestra PDA, es imposible almacenar el archivo de la misma forma en que aparece en la pantalla pública. Por esta razón, al momento de copiar la información, se examina el contenido del archivo y se realizan algunos ajustes (adaptaciones) para que la información pueda ser desplegada en la PDA.

IV.2.5 Facilidad de manejo de la información

En algunos dispositivos el espacio de trabajo para manejar la información está limitado a las dimensiones y tamaño que posee el dispositivo. Esto origina que el manejo de la información se dificulte o sea laborioso al usuario. Al presentarse esta problemática, el usuario se desvía de su objetivo puesto que, tiene que realizar una serie de cambios y ajustes necesarios para poder manejar la información. Además, en algunos dispositivos se dificulta la captura de los datos de entrada. Debido a esto, se plantea la utilización de pantallas públicas que provean un mayor espacio de trabajo para el usuario.

IV.3 Escenarios detallados

A partir de los ejemplos de aplicaciones mencionados anteriormente, se analizan y definen una serie de escenarios en los cuales se ve involucrada la participación del componente de migración. Dichos escenarios proporcionan una descripción minuciosa de situaciones hipotéticas bajo las cuales podría utilizarse el componente. En los siguientes escenarios se describe paso a paso la situación bajo la cual se presenta la migración de información.

Escenario 1: Planeación de la agenda y preparación de reunión

El usuario (quien está en su oficina sentado frente a su computadora) se encuentra preparando la agenda de la próxima reunión a realizarse. Por esta razón, crea una actividad en su agenda donde establece fecha, hora, lugar, ponentes y participantes para la reunión. Una vez que ha establecido en la agenda los puntos a tratar en la reunión, el usuario asigna las ligas al material de apoyo (documentos, presentaciones, videos, fotos, etc.) a cada uno de los diferentes puntos. Al finalizar la asignación del material, el usuario almacena la actividad de la agenda como un archivo. Para llevar a cabo esta tarea, el usuario debe: nombrar el archivo y seleccionar un directorio específico en donde desea almacenarlo. Ya definido el nombre y lugar de almacenamiento, el usuario guarda el archivo de la agenda y el material de apoyo asociado.

Al momento de efectuar la reunión, el usuario camina hacia la sala de reuniones, lugar donde se va a llevar a cabo la reunión. Adentro de la sala, el usuario desea proyectar la agenda y, por consecuencia, el material asociado a la agenda en una pantalla pública. Para esto, desea utilizar el proyector disponible en la sala. Para obtener el acceso al archivo de la agenda y al material asociado, el usuario debe autentificarse dentro del sistema. En el caso de que se verifique correctamente el acceso del usuario se realizan los siguientes puntos: 1) se transfiere la información del archivo a la pantalla pública, 2) se determina la aplicación asociada al archivo, 3) se proyecta la información en la pantalla.

Una vez desplegada la información en la pantalla, el usuario puede interactuar con dicha información para llevar a cabo la presentación del material que se encuentra ligado a los puntos a tratar en la agenda.

Escenario 2: Envío de información en base a la ubicación del usuario

Los miembros de una familia acaban de llegar a la ciudad y se encuentran perdidos tratando de llegar al hotel en el cual van a hospedarse. Por esta razón, el padre de familia decide utilizar su PDA y buscar en Internet información referente al hotel. Al revisar la información que provee el sitio del hotel, se encuentra con que éste cuenta con un sistema de navegación. Mediante dicho sistema, se proporcionan las instrucciones de cómo llegar al hotel a partir de la ubicación donde estos se encuentren. Por tal motivo, el padre de familia decide bajar e instalar el sistema. Una vez que completa la instalación del sistema en su PDA, se decide a abrir el sistema de navegación. Al abrir la aplicación, el sistema determina su ubicación. En base a dicha ubicación, se estima la ruta que deben seguir para llegar al hotel. Dicha ruta, es desplegada en la PDA en forma de mapa el cual contiene la información necesaria para que la familia pueda llegar a su destino a partir de la ubicación en donde se encuentran.

Escenario 3: Colaboración en la revisión de documentos

Un usuario se encuentra trabajando en la redacción de un artículo. Cuando termina de escribirlo, decide enviarlo a su co-autor para que lo revise. Antes de enviar la información, el autor establece ciertas restricciones que le permiten llevar un control sobre el manejo y contenido del documento. Cuando el co-autor recibe el archivo, éste se encuentra utilizando su PDA. Por esta razón, decide abrir el documento en la pantalla pública más cercana. De esta forma, el co-autor transfiere la información de la PDA a la pantalla. Una vez que la información ha sido transferida, se ejecuta la aplicación asociada al archivo para poder visualizar el contenido del mismo.

Ya desplegada la información en la pantalla, el co-autor analiza la información y escribe sus observaciones. Al momento de realizarlas se da cuenta que debido a las características propias del dispositivo que esta utilizando (de tipo público) solamente tiene derecho de lectura. Debido a esto, decide cambiarse a otro dispositivo. Por tal motivo, el co-autor cierra el archivo en la pantalla. Por lo tanto, la información que estaba publicada en la pantalla se pierde al momento de cerrar el archivo, por lo que ningún usuario que utilice la pantalla podrá acceder a ella.

El co-autor se dirige hacia su oficina para realizar la revisión del documento. Dentro de su oficina transfiere la información de su PDA a la laptop que ahí se encuentra. Una vez que la información es transferida a la laptop, se ejecuta la aplicación asociada al archivo y, de esta forma es posible visualizar la información. Puesto que el co-autor ya efectuó el análisis de la información, decide realizar directamente sus comentarios. Esta vez, dadas las características del dispositivo que está utilizando (de tipo personal), sí se le permite editar el documento. Por tal motivo, se le permite editar y almacenar la información en su laptop.

Al finalizar de realizar las observaciones al documento, el co-autor envía dicho documento al autor. El autor recibe el documento con las modificaciones efectuadas y analiza los cambios sugeridos por el co-autor.

IV.4 Servicios asociados al componente

En base a los escenarios planteados anteriormente, se realizó el análisis pertinente de la información. Dicho análisis permitió identificar el servicio al cual dará soporte el componente de migración: la transferencia de información. En la figura 22 puede observarse la representación del componente de migración, así como del servicio que este proporciona.

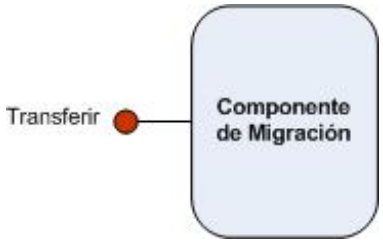


Figura 22. Componente de migración y servicios.

IV.4.1 Transferencia de información

La transferencia de información se refiere al servicio que proporciona el componente de migración para enviar información entre diversos dispositivos que pueden ser heterogéneos. Con esto, se busca el intercambio de información de un dispositivo fuente (laptop, PDA, PC, pantalla pública) a un dispositivo destino (pantalla pública, impresora, cañón, PDA, PC).

La información que se desee transferir debe estar almacenada físicamente en el dispositivo origen. Por tal motivo, la información a transferir deberá ser enviada por completo hacia el dispositivo destino. En el caso de transferencia de páginas web (URL), la única información que se transfiere es la liga de la página que se desea transferir.

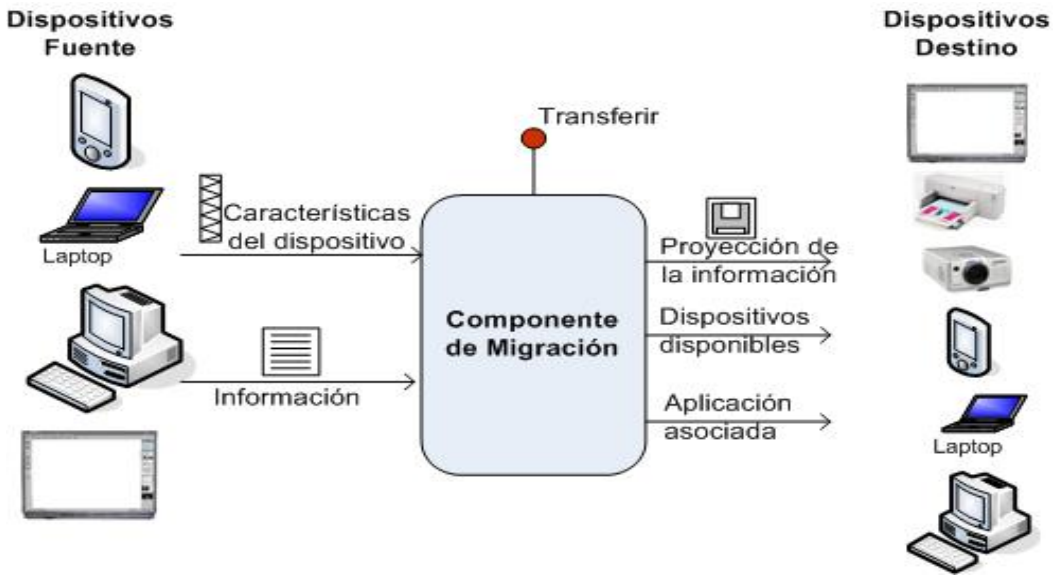


Figura 23. Transferencia de información.

En la figura 23, se puede observar que el servicio de transferencia consta de dos elementos principales: dispositivo fuente y dispositivo destino. Dichos elementos constan de una serie de valores de entrada y salida, los cuales son necesarios para permitir la transferencia de información entre ambos dispositivos.

IV.5 Especificación de requerimientos

El objetivo de esta especificación de requerimientos es definir de manera clara y precisa la funcionalidad y limitantes del componente a construir. En dicha especificación se establecen las normas o reglas básicas que el componente de migración debe cumplir y la funcionalidad a la que debe dar soporte. A continuación, se definen las restricciones, suposiciones y dependencias, y requerimientos [IEEE 830-1998] que deben considerarse en el desarrollo del componente:

IV.5.1 Restricciones

R1. La arquitectura del componente debe ser diseñada pensando en la reutilización. Es decir, el diseño de la arquitectura debe mostrar funcionalidad genérica y específica, que permita integrar diversos dispositivos sin importar cual es su naturaleza.

R2. La arquitectura del componente debe asegurar su extensibilidad. Es decir, la funcionalidad base provista por el componente puede ser extendida a partir del análisis de la arquitectura. De esta forma, se permite extender la arquitectura del componente de acuerdo a las necesidades de cada aplicación.

IV.5.2 Suposiciones y dependencias

S1. El componente será ejecutado en las siguientes plataformas: Windows XP, Windows CE.

S2. Los dispositivos bajo los cuales operará el componente son: PC, laptops, pantallas públicas (en particular Smart Boards), y PDA (Pocket PC).

D1. El desarrollo de este componente tomará como base los servicios que proporciona el middleware de SALSA.

IV.5.3 Requerimientos

REQ1. *Soporte a múltiples dispositivos heterogéneos*

Debido a la naturaleza de las aplicaciones o sistemas de cómputo ubicuo, el componente de migración deberá ser capaz de manejar ambientes que contengan diversos dispositivos heterogéneos. Es decir, en la migración de información deberá darse soporte a la participación de varios dispositivos, independientemente de su tipo.

REQ2. *Interacción transparente entre los diversos dispositivos*

Los distintos dispositivos que participen dentro del componente de migración deberán interactuar de forma transparente tanto para el usuario, como para el desarrollador que realiza las aplicaciones. Es decir, un usuario podrá interactuar fácilmente con el componente, sin conocer la lógica de la aplicación. Y, a su vez, el

desarrollador deberá ser capaz de desarrollar aplicaciones sin entender detalles de la forma en que los elementos de ésta interactúan.

REQ3. *Consciencia de la información contextual y ambiental*

El componente de migración deberá ser consciente de la información del contexto. Esto con la finalidad de poder dar respuesta a cambios que demande el desempeño de la aplicación.

REQ4. *Configuración del componente*

Se deberán proporcionar los mecanismos necesarios para poder instalar y configurar los parámetros de entrada que requiere el componente para su funcionamiento.

REQ5. *Adaptabilidad en la presentación de la información*

Se deberá poder adaptar el contenido de la información para que esta pueda ser visualizada de forma similar en los distintos dispositivos.

IV.6 Definición de entradas y salidas

A partir de la información obtenida en los puntos anteriores, se definieron las entradas y/o salidas necesarias para el funcionamiento del componente de migración. En la figura 24 pueden observarse dichas entradas y salidas.

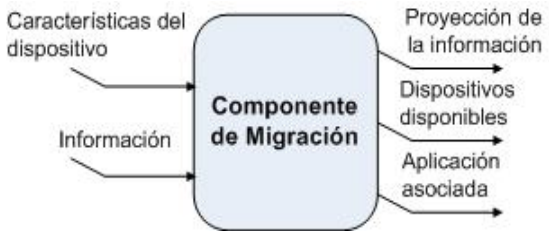


Figura 24. Componente de migración y sus entradas y/o salidas.

Cada una de estas entradas y salidas tiene un valor asociado, el cual es necesario para que dicho componente pueda funcionar. A continuación, se describen las entradas y salidas del componente de migración.

- ***Características del dispositivo:*** esta entrada se refiere a las restricciones propias tanto del dispositivo origen como del dispositivo destino. Las características que pueden establecerse son las siguientes:

    a) Tipo de dispositivo:

        - Público.
        - Personal.

    b) Limitantes de almacenamiento.

    c) Aplicaciones que provee.

- ***Información:*** se refiere a los datos que van a ser transferidos.
- ***Proyección de la información:*** si la información transferida será visualizada en el dispositivo de salida.
- ***Aplicación asociada:*** se refiere a la aplicación que puede ser utilizada para abrir y visualizar la información.
- ***Dispositivos disponibles:*** denota los dispositivos de destino disponibles a los cuales puede acceder el usuario para transferir la información.

IV.7 Casos de uso

Los casos de uso permiten obtener mayor información acerca de la funcionalidad del sistema. Además, complementan la información obtenida en la especificación de requerimientos. Para el componente de migración, se definieron una serie de casos de uso, los cuales permiten llevar a cabo la transferencia de información entre diversos dispositivos. En la figura 25 se muestra el diagrama de casos de uso de dicho componente.

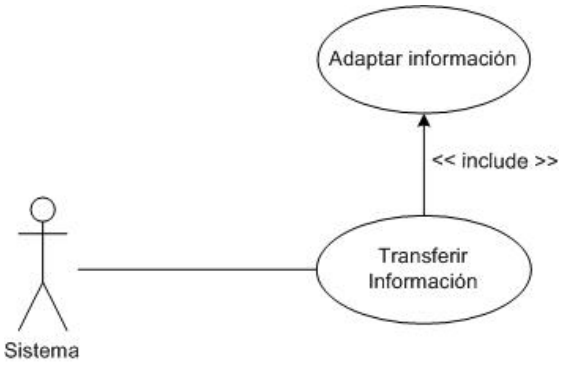


Figura 25. Diagrama de casos de uso para el componente de migración.

A continuación, se presenta la descripción del flujo de eventos de cada uno de los casos de uso que forman parte del componente de migración.

Caso de Uso: Transferir Información.

Actores: Sistema.

Objetivo: Trasladar información de un dispositivo a otro.

Curso típico de eventos:

Acción del actor	***Respuesta del sistema***
1. Este caso de uso inicia cuando el usuario desea transferir información de un dispositivo a otro. Para esto, la información a transferir se encuentra en el dispositivo que lleva consigo en esos momentos. 2. El usuario selecciona en su dispositivo la información que desea transferir. 3. El usuario selecciona el dispositivo destino al cual desea transferir el archivo.	4. El sistema recibe la información asociada (dispositivo destino, información, condiciones) a la transferencia de información. 5. El sistema determina si la información a transferir requiere de adaptación. Si la información requiere algún cambio, se inicializa el caso de uso *Adaptar información.* 6. El sistema envía la información del dispositivo fuente hacia el dispositivo destino (1). 7. El sistema determina el tipo de aplicación necesaria para visualizar el archivo (2). 8. El sistema proyecta la información del archivo (3). 9. El sistema envía la confirmación de la migración de información al usuario (3).

Cursos Alternos:

(1) En el caso de que el dispositivo destino no tenga la capacidad suficiente para recibir la información, el sistema envía un mensaje al dispositivo origen para notificarle que no es posible realizar la transferencia de la información.

(2) La información también podría enviarse a un proyector de video, entre otros dispositivos. Sin embargo, estos no se contemplan en este trabajo.

(3) Si el dispositivo destino es una impresora, la información se envía a impresión.

Pre-condiciones:

1. Requiere la autentificación del usuario en los dispositivos (origen y destino).

Caso de Uso: Adaptar información.

Actores: Sistema.

Objetivo: Realizar modificaciones en el contenido de la información para que esta pueda ser visualizada en el dispositivo destino.

Incluido en: Transferir información.

Curso típico de eventos:

Acción del actor	*Respuesta del sistema*
1. Este caso de uso inicia una vez que se verifican los privilegios relacionados con los dispositivos origen y destino.	2. El sistema recibe los privilegios de los dispositivos. 3. El sistema analiza la información y determina si deben efectuarse cambios. 4. Si se requieren cambios, el sistema determina que información puede adaptar para que esta pueda ser desplegada en el dispositivo destino (1). 5. El sistema realiza los cambios.

Cursos Alternos:

(1) Si no se requieren cambios, termina el proceso de adaptación y se continúa con la transferencia de la información.

Pre-condiciones:

1. La adaptación de información aplica cuando el dispositivo destino, posee una capacidad menor al dispositivo origen, o bien, como resultado de las restricciones establecidas por el usuario. Por ejemplo: la información desplegada en una pantalla pública puede incluir imágenes y texto. Sin embargo, si dicha información se desea transferir a una PDA, tal vez sería necesario suplir las imágenes por iconos más pequeños, que faciliten y agilicen, la transferencia de la información. Además, el PDA puede no tener la suficiente capacidad de memoria para recibir la información tal cual como se encuentre en el dispositivo de entrada.
2. Se requiere que el usuario establezca los privilegios del dispositivo para poder determinar si se requiere o no la adaptación.

IV.8 Interacción con otros componentes

Una de las principales características de los componentes de software es la composición con otros componentes. Esto significa que el componente debe ser lo suficientemente autocontenido y debe proveer una clara especificación de lo qué requiere y qué es lo que proporciona. En otras palabras el componente necesita encapsular su implementación e interactuar con su ambiente a través de interfaces bien definidas [Szyperski et al., 2002].

A partir de esta definición y del análisis de los escenarios planteados anteriormente, se pueden identificar tres componentes principales con los cuales interactúa el componente de migración: agenda, autentificación y localización. Dichos componentes, interactúan por medio de las interfaces y receptáculos que estos proporcionan. En la figura 26, podemos observar la forma en que interactúan los componentes.

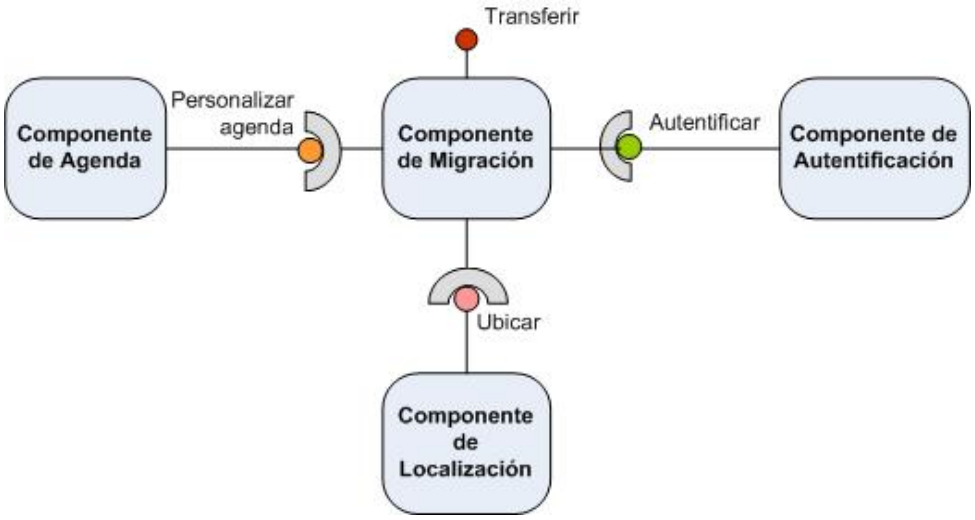


Figura 26. Interacción del componente de migración con otros componentes.

Los elementos presentados en la figura muestran las distintas interfaces que proveen los componentes. Mediante dichas interfaces el componente de migración puede acceder a los servicios que estos ofrecen. A continuación se describe la interacción entre los componentes:

- ***Interacción entre componente de: agenda y migración***

El componente de migración podrá interactuar con el componente de agenda a través de la interfaz de Personalizar agenda. El objetivo de esta interacción es permitir la transferencia de información personalizada. Para esto, el componente de migración requiere la información que contiene la agenda para adaptar la información que va a ser transferida. De esta forma, dicha información es personalizada de acuerdo a las actividades marcadas en la agenda del usuario.

- ***Interacción entre componente de: autentificación y migración***

En esta interacción se busca realizar la autentificación del dispositivo previamente a la transferencia de la información. En este caso, se plantea que antes de realizar la transferencia de la información ambos dispositivos (origen y destino) sean identificados para determinar si se encuentran en el mismo ambiente, o bien, ambos posean los privilegios necesarios para poder interactuar entre si.

- ***Interacción entre componente de: localización y migración***

La localización de servicios y usuarios es un elemento muy importante dentro del cómputo ubicuo. Por esta razón, y para facilitar la transferencia de la información, se plantea que el componente de localización (el cual determina la ubicación de usuarios y servicios) interactúe con el componente de migración para permitir la transferencia de información en base a la localización de usuarios y servicios.

IV.9 Resumen

En este capítulo presentamos el análisis del componente de migración. Para iniciar, se mostraron algunos ejemplos de aplicaciones en donde dicho componente pudiera ser utilizado. Partiendo de estas aplicaciones, se definieron detalladamente algunos escenarios de uso. En base a esto, se definió el servicio asociado al componente. Continuando con el análisis se establecieron los requerimientos y se definieron las entradas y salidas que debe cumplir el sistema. En base a esta información, se elaboraron los casos de uso del sistema. Finalmente, se definió la interacción del componente de migración con otros componentes.

En el siguiente capítulo se describe el diseño del componente de migración, arquitectura, diagramas de secuencia, diagramas de clases y modelo del componente. Para finalizar, se describe la implementación del componente.

Capítulo V

Diseño e implementación del componente de migración

V.1 Introducción

En este capítulo se presenta el diseño e implementación del componente de migración. El primer punto a desarrollar es el diseño de la arquitectura. Dicha arquitectura muestra los elementos necesarios que requiere el componente y como estos se relacionan.

A partir de esta arquitectura, se puede definir de forma abstracta el funcionamiento del componente. Esto facilita el entendimiento de la funcionalidad que provee dicho componente. Continuando con el diseño, el modelo del componente muestra los distintos elementos que en su conjunto representan al componente. De esta forma, se establece claramente porqué es un componente.

Los diagramas de secuencia muestran el flujo de acciones necesarias para llevar a cabo la migración de información. Estos diagramas fueron desarrollados en dos niveles: a nivel general, define de forma genérica el flujo de acciones; y a nivel detallado, a partir de un escenario específico se describe el flujo de acciones necesarias para implementarlo.

En cuanto a la implementación del componente, primeramente se describen las tecnologías de desarrollo utilizadas en su implementación. Seguido, se hace una descripción de cómo funciona el descubrimiento de servicios. Para finalizar, se presenta el prototipo del componente en sus dos implementaciones: para dispositivos de bolsillo (PDA) y para dispositivos de escritorio (laptop, PC y pantallas públicas).

Las siguientes secciones, describen cada uno de los puntos citados anteriormente. El conjunto de esta información representa el diseño e implementación del componente de migración.

V.2 Arquitectura

El objetivo de la arquitectura es determinar o definir los elementos principales que sirven como base para realizar el proceso de migración de información. En este trabajo, el diseño de la arquitectura se realiza en dos niveles: El 1er. nivel, muestra una arquitectura abstracta; el 2do. nivel, presenta una arquitectura detallada o extendida.

En las siguientes secciones se describe detalladamente cada uno de los niveles de la arquitectura del componente de migración.

V.2.1 Arquitectura abstracta

La arquitectura abstracta presenta la estructura del sistema a un mayor nivel de abstracción. Esta arquitectura busca definir un esquema genérico que pueda ser entendido fácilmente y que encapsule el proceso de migración de información. La figura 27, muestra la arquitectura abstracta definida para la migración de información entre un dispositivo fuente y un dispositivo destino.

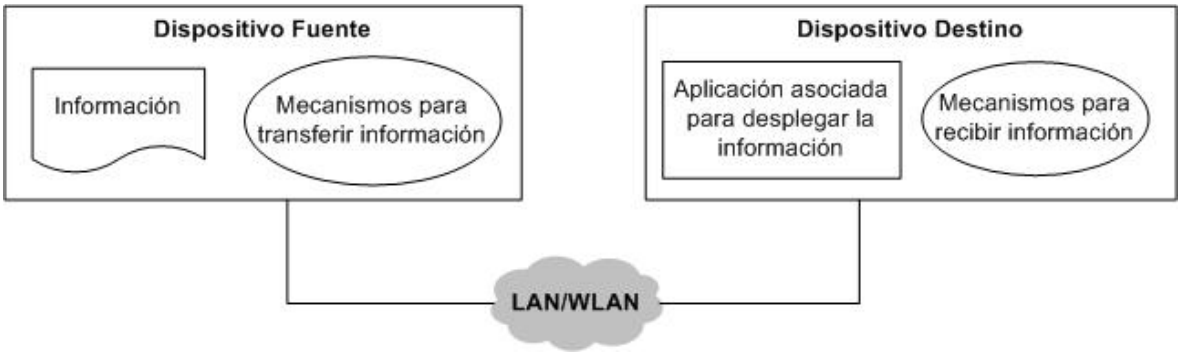


Figura 27. Arquitectura abstracta definida para la migración de información entre dispositivos.

En base a esta abstracción, se define una arquitectura basada en la funcionalidad que provee el middleware de SALSA. La figura 28 presenta los servicios o elementos principales a los cuales dará soporte el componente de migración: transferencia y

adaptación de la información. Estos elementos podrán interactuar entre sí para llevar a cabo la migración de información mediante los mecanismos de comunicación que provee el middleware de SALSA.

Figura 28. Arquitectura del componente de migración.

Dado que, se plantea representar a los diversos dispositivos como agentes que trabajen bajo el middleware de SALSA (ver figura 29), también es posible utilizar los mecanismos de comunicación que éste provee. De esta forma, los dispositivos (representados como agentes) podrán comunicarse entre ellos mediante los servicios básicos de comunicación que proporciona SALSA.

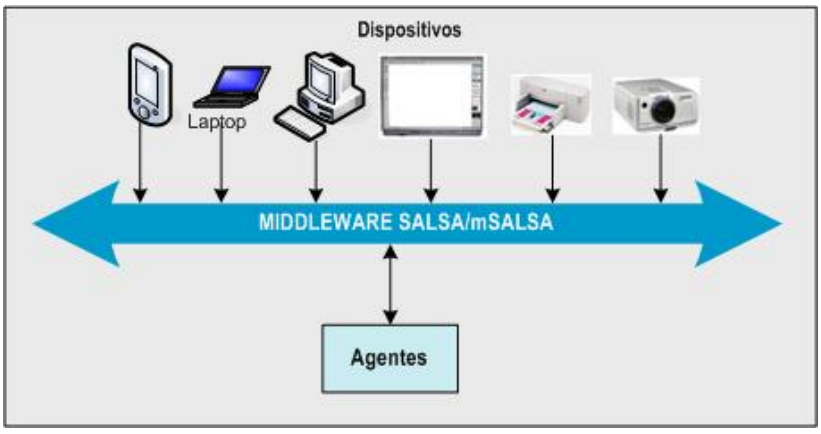


Figura 29. Representación de los dispositivos como agentes.

La figura 29, ilustra la representación de los diversos dispositivos como agentes en el middleware SALSA. Dichos dispositivos pueden encontrarse inmersos en el ambiente y llevar a cabo la migración de información mediante el uso de agentes.

A continuación, se describe detalladamente la arquitectura extendida para el componente de migración.

V.2.2 Arquitectura extendida

La arquitectura extendida presenta la estructura del sistema a un nivel más detallado. Esta arquitectura busca definir un esquema más específico para el proceso de migración de información. La figura 30 muestra la arquitectura extendida definida para la migración de información entre un dispositivo fuente y un dispositivo destino.

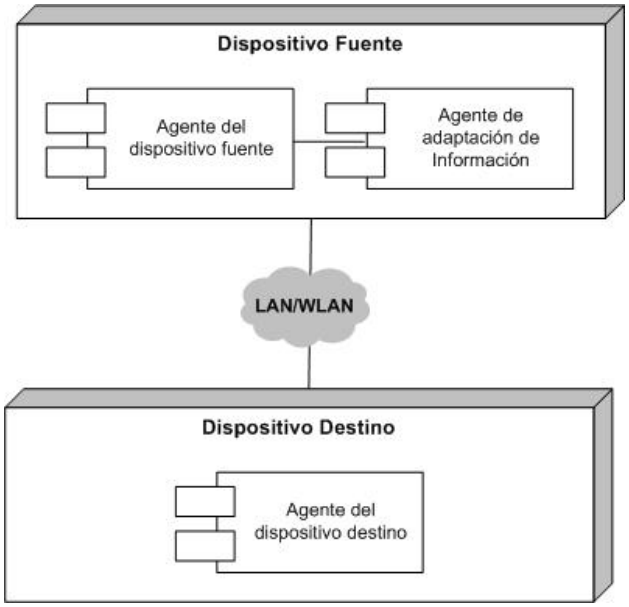


Figura 30. Arquitectura extendida para el componente de migración

En la figura 30, se presentan los elementos del componente, los cuales son descritos a continuación:

Agente del dispositivo fuente: este agente representa la información (ya sea el archivo o la referencia a la información) que el usuario desea transferir a otro dispositivo. Así como las características o privilegios (tipo de dispositivo, capacidad, etc.) que posee el dispositivo

fuente para permitir la migración de información, y los mecanismos necesarios para transferir la información.

Agente de adaptación de información: este agente se encarga de realizar las modificaciones que debe sufrir la información dadas las características que presenta el dispositivo destino y las especificaciones definidas por el dispositivo fuente.

Agente del dispositivo destino: este agente representa: las características (capacidad, tipos de aplicaciones, etc.) que posee el dispositivo para recibir información; al dispositivo mismo el cual será capaz de aceptar o rechazar la solicitud de transferencia; los mecanismos necesarios para recibir la información.

Todas las características e información relacionada con el control y la publicación de información, es manejada a través del estudio de privacidad que realiza [Tentori, 2005]. El diseño e implementación de estos elementos puede consultarse en el apéndice B.

V.3 Diseño

El diseño del componente de migración inicia con la definición de un modelo abstracto de la funcionalidad del componente. Donde, a partir de dicha abstracción se definen interfaces y el modelo mismo del componente.

En base a dicha información, se elaboran el diagrama de clases y los diagramas de secuencia necesarios para definir la funcionalidad que provee el componente.

Las siguientes secciones presentan el diseño de los elementos que forman parte del componente de migración.

V.3.1 Abstracción de la funcionalidad del componente

La migración de información se define como el proceso que se lleva a cabo entre dos dispositivos para transferir información de uno a otro. El proceso para migrar información de un dispositivo A (denominado dispositivo fuente) a un dispositivo B (denominado dispositivo destino) es el siguiente (ver figura 31):

1. El dispositivo fuente (dispositivo A) solicita al dispositivo destino (dispositivo B) el envío de la información.
2. El dispositivo destino recibe y analiza la solicitud para determinar si acepta o no la transferencia de la información.
3. Si el dispositivo destino acepta la transferencia de información, se notifica al dispositivo fuente que se aceptó la transferencia.
4. El dispositivo fuente envía la información al dispositivo destino.
5. El dispositivo destino recibe la información y notifica al dispositivo fuente que recibió la información.
6. El dispositivo destino determina el tipo de aplicación asociada a la información.
7. Si no existe una aplicación asociada a la información en el dispositivo destino, se despliega un listado de aplicaciones al usuario para que seleccione una.
8. El usuario selecciona la aplicación con la cual desea abrir la información.
9. El usuario envía el resultado al dispositivo destino.
10. El dispositivo destino despliega la información con la aplicación seleccionada por el usuario.

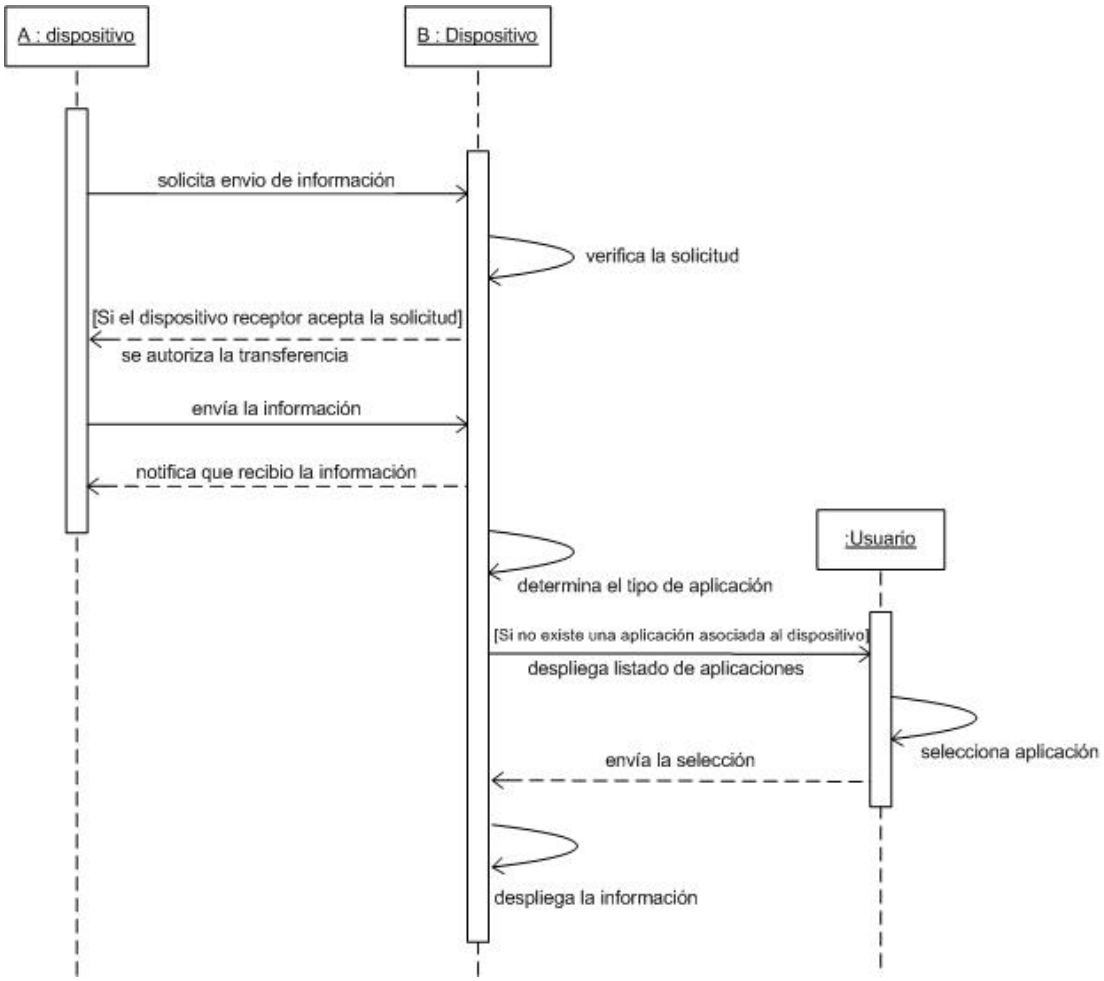


Figura 31. Abstracción de la migración de información entre dos dispositivos.

La figura 31 muestra la secuencia de acciones necesarias para describir de manera abstracta la migración de información entre un dispositivo fuente y un dispositivo destino.

V.3.2 Modelo del componente

El modelo del componente está compuesto de varios elementos los cuales de manera conjunta definen los estándares para [Supriya, 2002]:

- *Definir Interfaces:* son aquellas interfaces que son soportadas por el componente.
- *Nombramiento de componentes e interfaces:* el componente y las interfaces deben poseer nombre representativos que permitan identificarlos fácilmente.

- *Metadatos:* es la información sobre los componentes, interfaces y sus relaciones, así como los APIs necesarios para acceder a dicha información.
- *Interoperabilidad entre componentes:* se denomina a la comunicación e intercambio de datos entre los componentes desarrollados por diversos desarrolladores o compañías utilizando plataformas o lenguajes de programación distintos.
- *Adecuación del componente*: facilidades para que un cliente adapte un componente antes de su instalación o uso.
- *Composición con terceros:* son reglas de combinación de componentes para crear grandes estructuras y para sustituir y agregar componentes a estructuras existentes.
- *Soporte a la evolución:* reglas y servicios para reemplazar componentes o interfaces por versiones más nuevas.
- *Empaquetamiento y ejecución:* empaquetamiento de la implementación y recursos necesarios para instalar y configurar un componente.

[Völter, 2002] retoma la definición presentada por [Szyperski et al., 2002] de la cual abstrae los principales elementos de un componente:

- Interfaces especificadas contractualmente.
- Dependencias explícitas del contexto.
- Ejecución independiente.
- Composición con terceros.

Donde cada unos de estos elementos agrupa un conjunto de elementos definidos para el modelo del componente. En base a esto, se decidió crear el modelo del componente de migración, el cual se compone de los siguientes elementos: definición de interfaces; dependencias explícitas; definición de la ejecución; y composición con terceros.

A continuación, se describe de manera detallada cada uno de dichos elementos dentro del contexto del componente de migración. El conjunto de estos representa el *modelo del componente de migración*.

V.3.2.1 Definición de interfaces

La definición de las interfaces que provee el componente es un punto muy importante, puesto que, los usuarios sólo pueden invocar operaciones definidas en dichas interfaces. De esta forma, los usuarios pueden interactuar con el componente para llevar a cabo la transferencia de información sin conocer la lógica de la implementación. En este caso en particular, la interfaz definida para el componente de migración permite la interacción a través de una lista. Dicha lista contiene el listado de los posibles dispositivos destino con los cuales puede interactuar para transferir la información. En la figura 32 se muestran los atributos y operaciones que ofrece el componente mediante la interfaz.

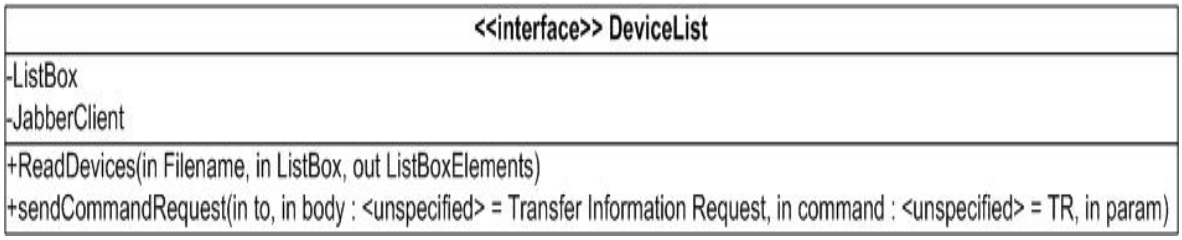


Figura 32. Especificación de la interfaz del componente.

A continuación, se describe cada uno de los elementos que forman parte de los atributos de la interfaz del componente:

- *ListBox*: atributo que denota la lista de los posibles dispositivos destino con los cuales puede interactuar el usuario para transferir la información.
- *JabberClient*: este atributo denota el cliente jabber a través del cual la interfaz envía la solicitud de transferencia al dispositivo destino.

Las operaciones que ofrece la interfaz del componente son las siguientes:

- *ReadDevices*: esta operación permite leer de un archivo XML la lista de los dispositivos destino. La generación de dicha lista es dinámica puesto que, debe

considerar únicamente a aquellos dispositivos que se encuentren disponibles en el ambiente al momento de ejecución. Por esta razón, los parámetros de entrada que recibe esta operación son: nombre del archivo (***Filename***), y lista (***ListBox***). Como salida, esta operación regresa los elementos que deben ser agregados a la lista.

- *sendCommandRequest*: mediante esta operación se realiza la solicitud de transferencia al dispositivo destino. Los parámetros de entrada que recibe son los siguientes:
    - *To*: denota el identificador del dispositivo destino a quien se solicita la transferencia de la información.
    - *Body*: es el título que lleva el mensaje que es enviado al dispositivo destino. En particular, en esta operación lleva por título: ***Transfer Information Request.***
    - *Command*: es el comando que se requiere en esta operación. En este caso en particular tiene el valor asignado de: ***TR.***
    - *Params*: son los parámetros adicionales que se envían en el mensaje para identificar la solicitud de transferencia. Los elementos definidos para este parámetro en esta interfaz son los siguientes:
        1. ***filename***: representa el nombre del archivo a transferir. Incluye la ruta en donde se encuentra almacenado.
        2. ***infotype***: tipo de la información a transferir. Los valores que puede tomar en esta versión del componente son:
            a. ***filetype***, se refiere a la transferencia de un archivo.
            b. ***url***, se refiere a la transferencia de un URL.
        3. ***name***: nombre del archivo a transferir (sin la ruta).
        4. ***size***: tamaño en Kb, del archivo que se desea transferir. En el caso de la transferencia de URL, el tamaño es igual a 1Kb.
        5. ***dtSource***: tipo del dispositivo origen que solicita la transferencia de información. Los tipos de origen pueden ser:

a. Pda.

b. Pc.

c. Laptop.

d. Display.

Como ejemplo, presentemos el siguiente mensaje enviado por la interfaz a través de la operación *SendCommandRequest*.

```
String enviar = "<filename>C:\\prueba\\prueba.doc</filename>"
      + "<infotype>file</infotype>"
      + "<to>display@pc-coolab9.cicese.mx</to>"
      + "<name>prueba.doc</name>"
      + "<size>36Kb</size>"
      + "<dtSource>pda</dtSource>";

sendCommandRequest(to, "Transfer Information Request","TR",
enviar);
```

V.3.2.2 Definición de dependencias explícitas

La definición de las dependencias explícitas del contexto busca establecer:

1) Las llamadas a operaciones del ciclo de vida que son utilizadas por el contexto del contenedor para controlar el ciclo de vida de la instancia del componente, por ejemplo:

   - Activación.
   - Creación.
   - Destrucción.
   - Pasivación.

2) Las anotaciones (descriptores de ejecución) que especifican lo que espera una instancia del componente de su contenedor, por ejemplo:

   - Servicios provistos por el contenedor (transacciones, seguridad, etc.)
   - Recursos disponibles en el ambiente (conexiones a bases de datos, colas de mensajes, etc.)

3) La especificación de interfaces (de otros componentes) requeridas para que el componente pueda ser ejecutado exitosamente.

4) La disponibilidad de interfaces en el contenedor al momento de ejecución.

A continuación, se describen cada uno de estos elementos, los cuales definen las dependencias explícitas del contexto para el componente de migración.

Llamadas a operaciones del ciclo de vida:

Las llamadas a operaciones definidas en el contexto del contenedor son referentes a la activación y destrucción de las instancias del componente. La figura 33, muestra las operaciones definidas para especificar las dependencias del contexto dentro del ciclo de vida del contenedor del componente.

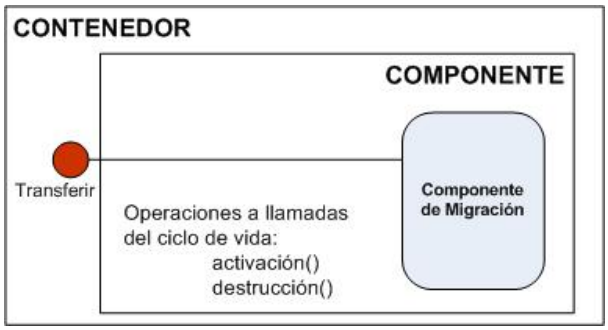


Figura 33. Llamadas a operaciones del ciclo de vida.

Las operaciones definidas en el contenedor del componente de migración son:

- *Activación:* se refiere a la creación y activación de las instancias del componente dentro del contenedor.
- *Destrucción:* se refiere a la destrucción de instancias del componente.

Anotaciones:

Las anotaciones especificadas para el componente de migración, residen en una parte especial del contenedor. En la figura 34, se muestran las anotaciones definidas para el componente.

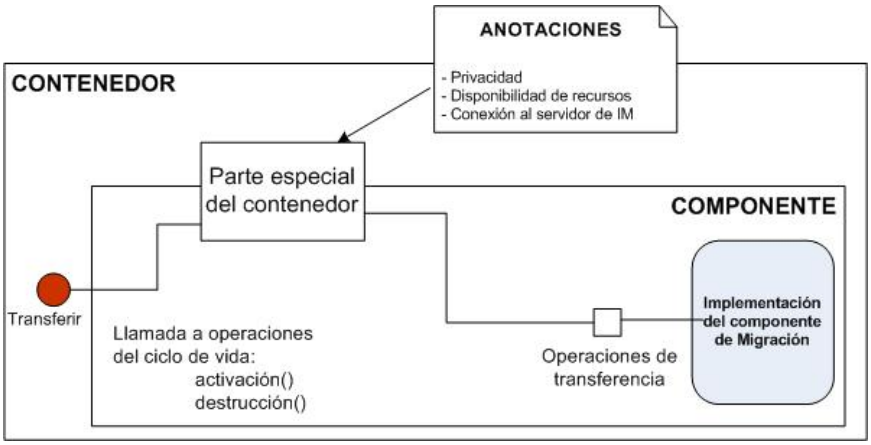


Figura 34. Anotaciones del contenedor.

Como puede observarse en la figura, las anotaciones definidas en el contenedor del componente de migración son:

- *Privacidad*, las anotaciones de privacidad hacen referencia a las características y capacidades de interacción entre los distintos dispositivos (instancias del componente) que forman parte del ambiente de ejecución del componente. [Tentori, 2005] presenta el estudio de privacidad para el componente de migración. Para mayor descripción de dicho estudio ver apéndice B.
- *Disponibilidad de recursos*, las anotaciones de disponibilidad de recursos se refieren a todos aquellos dispositivos que se encuentran disponibles en el ambiente, al momento de ejecución del componente.
- *Conexión al servidor de mensajería instantánea*, las anotaciones de conexión se refieren a la validación de la conexión al servidor de mensajería instantánea, ya que dicho servidor es el encargado de realizar toda la parte de comunicación entre las instancias del componente.

Interfaces requeridas:

La especificación de interfaces requeridas para que el componente de migración pueda interactuar con otros componentes es presentada en la figura 35.

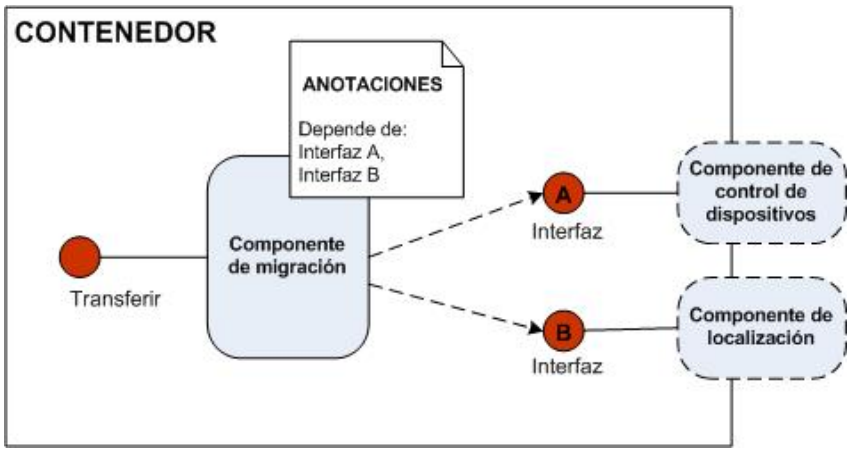


Figura 35. Interfaces requeridas para la interacción con otros componentes.

Como puede observarse en dicha figura, los interfaces que debe proveer el componente son:

- Interfaz A, esta interfaz especifica las operaciones necesarias para que el componente de migración pueda interactuar y ser ejecutado en colaboración con el componente de control de dispositivos presentado por [Markarian, 2005].
- Interfaz B, define las operaciones requeridas para que logre efectuarse la colaboración entre el componente de migración y el componente de localización.

Disponibilidad de interfaces:

La disponibilidad de interfaces se refiere a la verificación de elementos que realiza el contenedor al momento de ejecutar una instancia del componente. Es decir, en esta parte, se verifica que existan todos los elementos necesarios y requeridos para llevar a cabo las operaciones definidas en las interfaces del componente de migración.

V.3.2.3 Definición de independencia de ejecución

En este punto se busca definir y establecer las condiciones necesarias para lograr la independencia de ejecución del componente. Los elementos que deben asegurarse en esta parte son:

1) *Instalación del componente*, dado que los componentes requieren un paso explícito de instalación que los habilite dentro de su contenedor.

2) *Dependencias del contexto*, dado que deben verificarse y asegurarse que se cumplan con las dependencias de contexto al momento de ejecución.

A continuación, se describen cada uno de estos elementos relacionados con el componente de migración.

Instalación del componente:

En la instalación del componente de migración se requiere asegurar la disponibilidad de la instancia creada del componente dentro del contenedor. De esta forma, existen ciertos elementos que son requeridos al momento de instalación, tales como:

- Anotaciones.
- Implementación.
- Interfaces.

En la figura 36, pueden observarse los elementos necesarios para la instalación del componente:

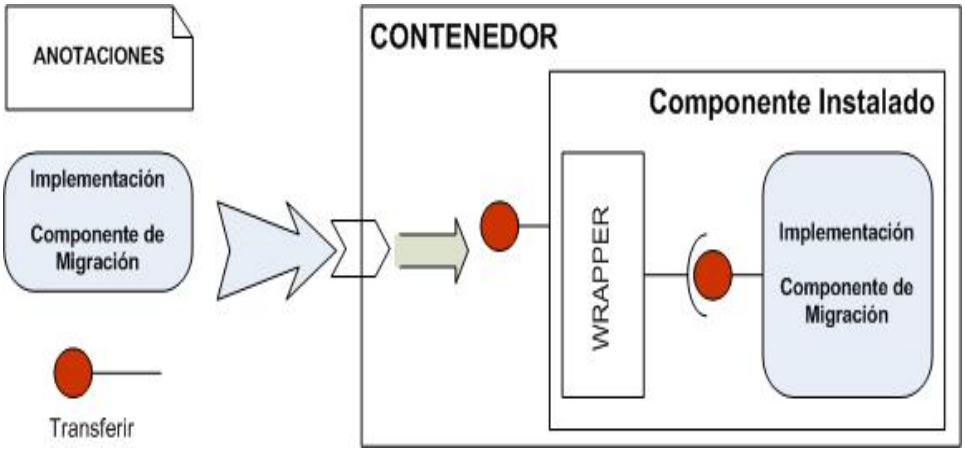


Figura 36. Elementos requeridos para la instalación del componente.

Una copia o instancia de dichos elementos es almacenada y utilizada por el contenedor para asegurar que el componente de migración pueda ser ejecutado de manera independiente por cada una de las instancias generadas para el componente.

Dependencias de contexto:

Al momento de ejecución de alguna instancia del componente debe verificarse que se cumplan con las dependencias del contexto definidas en las anotaciones del componente. Realizar esta verificación y comprobar la disponibilidad de los elementos requeridos para la ejecución del componente, permite asegurar la independencia de ejecución. Ya que, de esta manera, se asegura el correcto funcionamiento del componente.

V.3.2.4 Definiendo la composición

Los aspectos que definen la composición del componente son los siguientes:

1) El desarrollo de cada uno de los puntos citados anteriormente facilita que el componente pueda ser utilizado y ensamblado en otras aplicaciones.

2) El proporcionar una forma controlada para la configuración del componente, utilizando parámetros y anotaciones.

3) Definir un mecanismo de empaquetamiento, que encapsule al componente.

De esta forma, podemos definir la composición del componente de migración puesto que:

- Se cumplen los requerimientos establecidos en el punto 1.
- Se provee de un mecanismo de configuración en el cual pueden ser modificados los parámetros y las anotaciones del componente.
- Se proporciona un mecanismo de empaquetamiento que contiene los elementos necesarios para instalar y utilizar el componente de migración. En la figura 37, se pueden observar los elementos que forman parte del paquete del componente de migración.

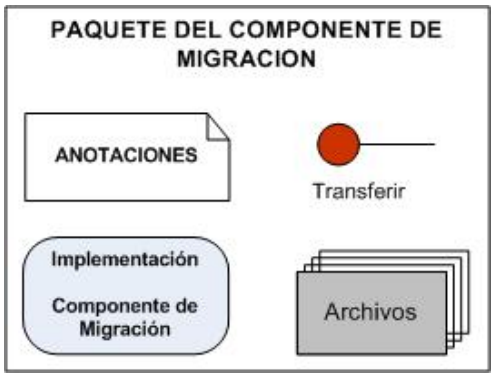


Figura 37. Paquete del componente de migración.

A partir de la información obtenida hasta este punto, pueden identificarse las distintas clases de las cuales se compone el componente de migración. En la siguiente sección se presenta el diagrama de clases del componente.

V.3.3 Diagrama de Clases

En esta sección se presenta el diagrama de clases del componente de migración. Dado que, el desarrollo del componente se basa en el middleware de SALSA, los elementos de dicho componente son representados como agentes. En la figura 38 se muestra el diagrama de clases para el componente de migración. Dicha figura muestra la representación de los distintos agentes involucrados en la transferencia de información, así como, los diversos elementos que los componen: razonamiento y acción.

El funcionamiento principal del sistema recae en el agente representado en la clase *DeviceAgent.* Esta clase se encarga de crear el agente e inicializar los valores del cliente Jabber. La clase *DeviceReasoning* recibe todos los mensajes que son enviados al agente y en base a estos mensajes se encarga de la toma de decisiones del agente. A través de la clase *DeviceList* el usuario puede interactuar con el agente para realizar la transferencia de información. Las clases heredadas de la clase *Action* se encargan de toda la lógica de la implementación para llevar a cabo la transferencia de la información. La clase *AdaptAgent* se encarga de la adaptación de la información. La clase *AdaptReasoning* determina los cambios o modificaciones que debe sufrir la información que va a ser transferida. La clase *AdaptFinished* actúa para notificar que la información fue adaptada.

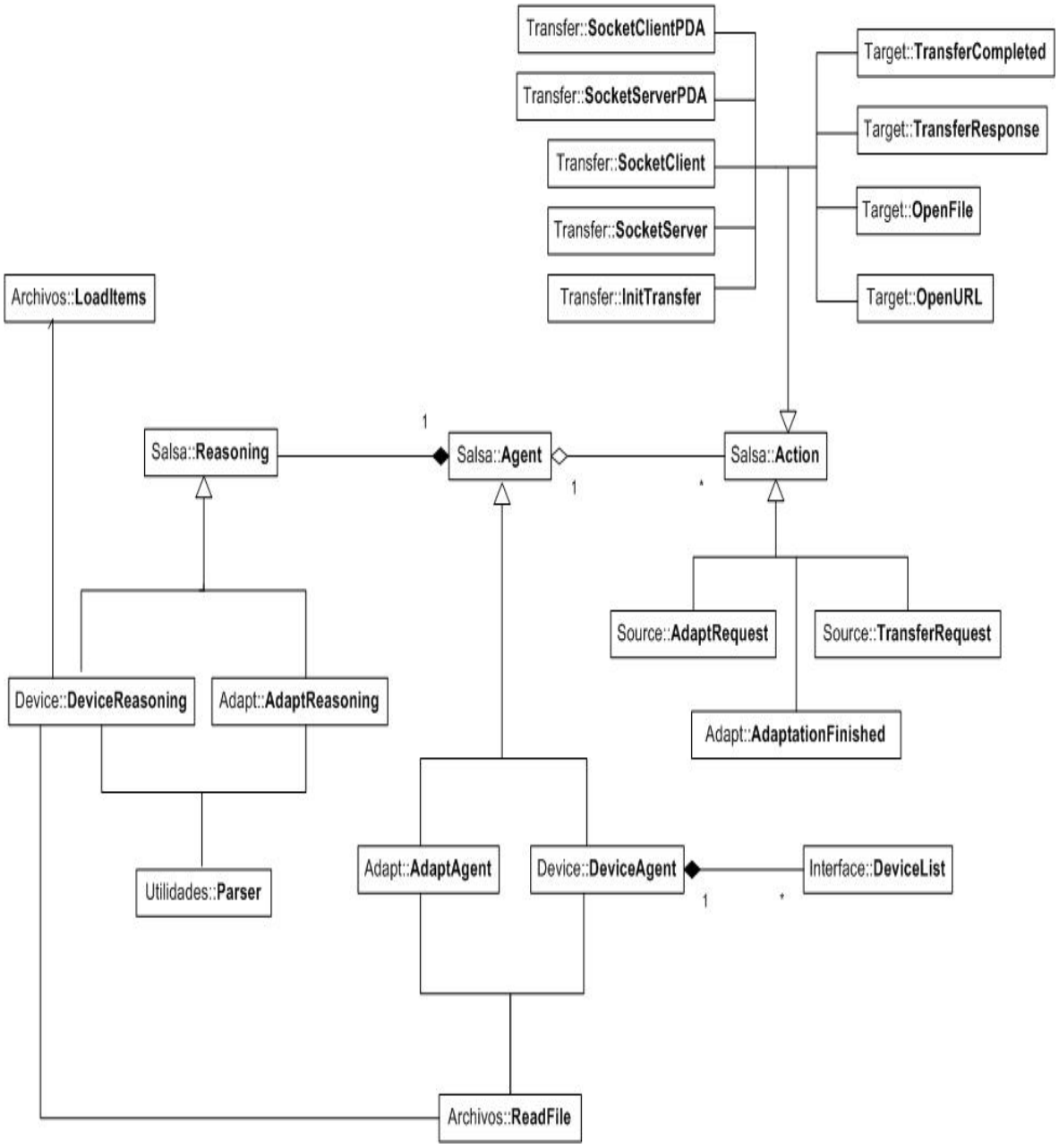


Figura 38. Diagrama de clases del componente de migración.

En la siguiente sección se muestran los diagramas de secuencia del componente. Dichos diagramas definen la secuencia de acciones necesarias para realizar la transferencia de información.

V.3.4 Diagramas de secuencia

Los diagramas de secuencia muestran el orden y la forma en la cual se llevan a cabo las acciones para lograr un objetivo. En particular, en esta sección se muestran los diagramas de secuencia necesarios para realizar la migración de información entre dos dispositivos.

La figura 39 presenta la secuencia de acciones necesarias para efectuar la transferencia de información entre un dispositivo fuente y un dispositivo destino.

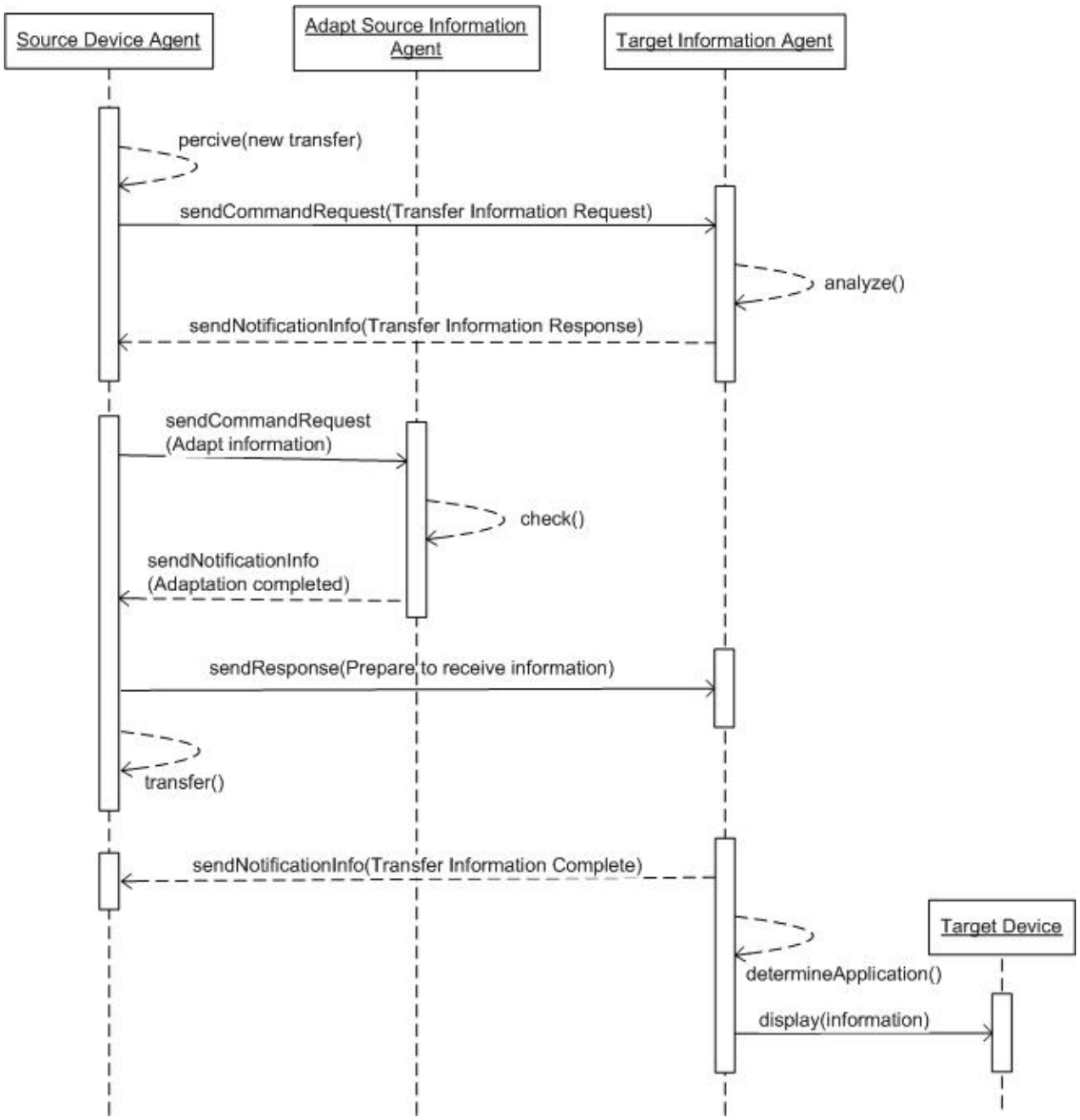


Figura 39. Diagrama de Secuencia para transferir la información.

En la figura 40, se muestra la secuencia de acciones que se llevan a cabo para adaptar la información que va a ser transferida. Esto, con la finalidad de reemplazar partes del contenido de la información a transferir.

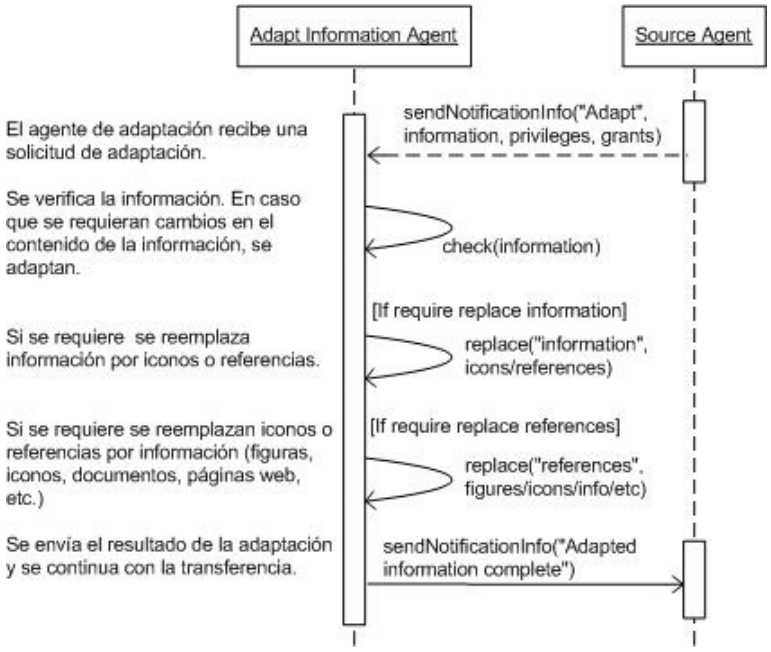


Figura 40. Diagrama de Secuencia para adaptar la información.

En base a la secuencia de acciones definidas en estos diagramas, se decidió profundizar el flujo de acciones. De esta forma, en la siguiente sección se presentan los diagramas de secuencia detallados para el componente de migración.

V.3.5 Diagramas de secuencia detallados

Los diagramas de secuencia detallados muestran el flujo de acciones necesarias para llevar a cabo un objetivo en particular. La base para el desarrollo de estos diagramas se toma del trabajo de tesis realizado por [Preciado, 2004], en donde se presenta el siguiente escenario:

> *Cuando el Dr. García examina al paciente en la cama 234, los resultados de rayos X que solicitó son incluidos en el expediente electrónico de su paciente en la cama 225. El agente de HIS le notifica esto al doctor mediante un mensaje enviado a través del agente broker. El Dr. García aprovecha la cercanía de una pantalla pública cuando termina la consulta al paciente actual y, antes de ver al paciente de*

la cama 225. La localización del doctor, la cual es constantemente detectada por el agente de la estimación de localización en su PDA, es notificada a todos los usuarios y agentes en su ambiente, tal como al agente consciente del contexto y al agente de la pantalla pública(PD-a). ***El PD-a reconoce la presencia del usuario desplegando su fotografía, esto indica que el usuario ha ingresado en el sistema, y las aplicaciones en la pantalla se personalizan para él***[1].

Además de su fotografía, al Dr. García se le muestran dos aplicaciones personalizadas, el PublicCalendar y el PublicMap. El PublicCalendar despliega la agenda de médicos y otros eventos públicos publicados por el hospital u otros miembros. Las entradas personalizadas son obtenidas desde el PDA del usuario a través del User's Proxy Agent. ***El médico puede arrastrar eventos públicos de su interés hacia su foto, esto transferirá dicha información a su agenda en su PDA***[2]***.***

Basándonos en el escenario ([2]), se determinó extender o detallar el flujo de acciones necesarias para realizar la transferencia de información.

La figura 41, presenta la secuencia de acciones determinadas para efectuar la migración de información entre una pantalla pública y una PDA. La secuencia que se define es la necesaria para realizar la transferencia de un URL de una pantalla pública a una PDA.

1) El agente que representa a la pantalla pública (Public Display Agent) percibe la solicitud de una nueva transferencia.
2) El agente de la pantalla solicita al agente del usuario (User's Proxy Agent) la transferencia del URL. El agente del usuario determina si acepta o no la solicitud.
3) En este caso, puesto que la información no requiere adaptación (se trata de un URL), se transfiere la información de un agente a otro.
4) El agente del usuario recibe el URL y lo manda al dispositivo para que este se encargue de desplegar la información en pantalla.
5) El agente del usuario notifica al agente de la pantalla que la transferencia se realizó satisfactoriamente.

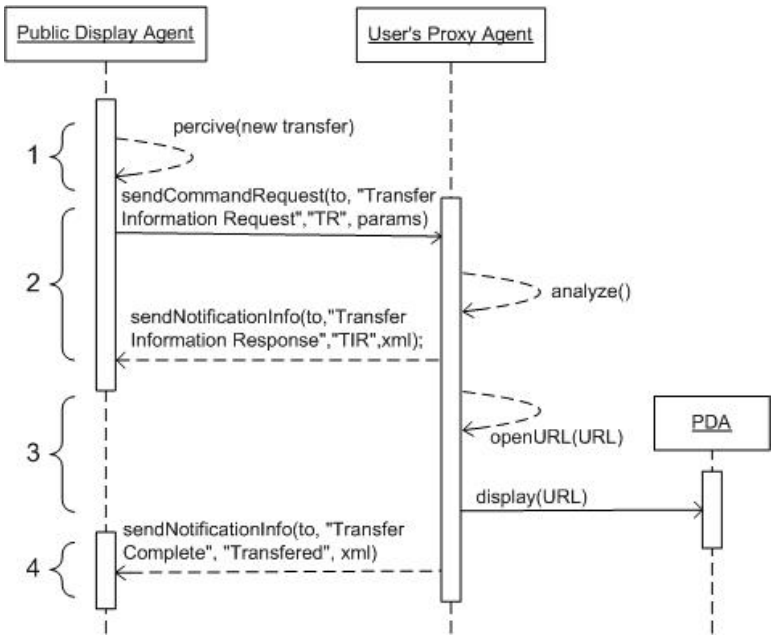


Figura 41. Diagrama de Secuencia detallado para la transferencia de un URL entre dos dispositivos.

Otro escenario en el cual podría presentarse la migración de información es el planteado en el escenario ([1]). En dicho escenario, el dispositivo destino solicita información al dispositivo fuente. Como caso particular, se partirá del siguiente escenario:

> *El usuario se acerca a una pantalla pública para abrir cierta información. El agente que representa a la pantalla pública detecta la presencia del usuario y le solicita la información necesaria para identificar de quien se trata. El agente que representa al usuario, envía al agente de la pantalla los datos necesarios para que éste lo identifique. Con estos datos, el agente de la pantalla identifica al usuario y muestra su fotografía en la pantalla pública.*
>
> *Al mismo tiempo, el agente de la pantalla solicita información adicional al agente del usuario para personalizar la aplicación del PublicCalendar. El agente del usuario envía la información requerida por el agente de la pantalla, el cual actualiza la información del PublicCalendar que se encuentra publicada en la pantalla pública.*

En base al escenario anterior, se determinó la secuencia de acciones necesarias para llevarlo a cabo. La figura 42, muestra el flujo de acciones que se realizan para lograr la personalización de la información en la pantalla pública. La secuencia que se define es la siguiente:

1) El agente que representa a la pantalla percibe la presencia de un nuevo usuario.
2) El agente de la pantalla identifica al usuario y despliega su fotografía en la pantalla pública.
3) El agente de la pantalla solicita información adicional para actualizar la aplicación del PublicCalendar.
4) El agente del usuario verifica la información y solicita la adaptación de la información.
5) El agente de adaptación modifica la información personal, reemplazándola por iconos. Una vez que realiza la adaptación, notifica al agente del usuario que se realizó la adaptación de la información.
6) El agente del usuario solicita al agente de la pantalla que se prepare para recibir la información.
7) El agente del usuario transfiere la información al agente de la pantalla.
8) Una vez que el agente de la pantalla recibe la información, notifica al agente del usuario que se realizó la transferencia de la información satisfactoriamente.
9) El agente de la pantalla personaliza la aplicación del PublicCalendar y despliega la información personalizada en la pantalla pública.

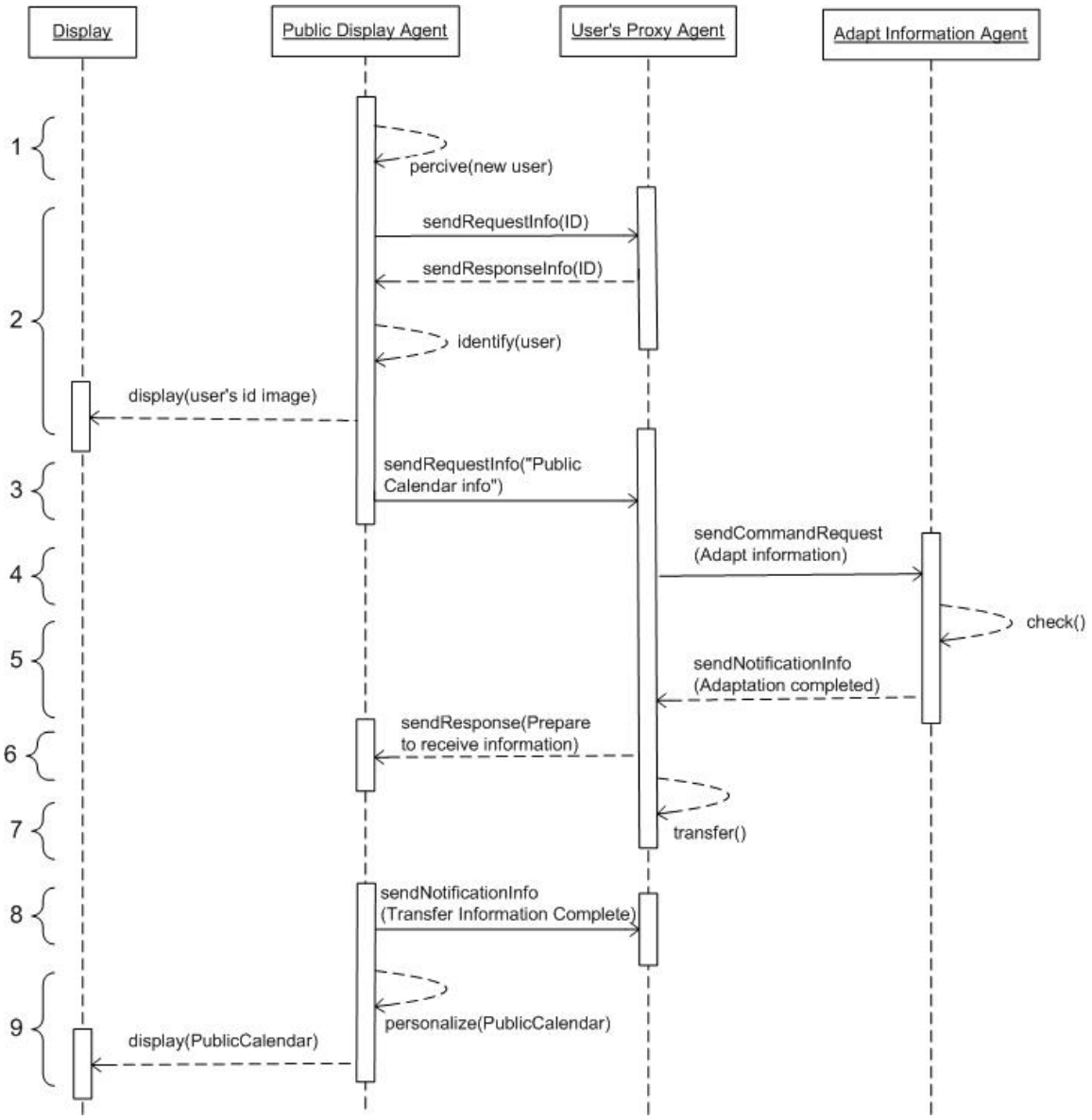


Figura 42. Diagrama de Secuencia detallado para personalizar la información del usuario.

Una vez definido el diseño del componente, se continúa con la implementación del mismo. En las siguientes secciones se muestra como se realizó la implementación del componente de migración.

V.4 Implementación del componente

El componente de migración posee dos implementaciones: una para dispositivos móviles (PDA) y otra para dispositivos de escritorio (PC, laptop, pantallas públicas). Dichas

implementaciones fueron desarrolladas en los lenguajes de programación C# y Java, respectivamente. Ambas implementaciones toman como base la funcionalidad que ofrece el middleware de SALSA.

A continuación, se describe a detalle la manera en la cual se realizaron las distintas implementaciones del componente de migración.

V.4.1 Tecnologías de desarrollo

Para el desarrollo del componente de migración se utilizan distintas tecnologías. En la figura 43 pueden apreciarse los elementos tecnológicos utilizados en la implementación del componente.

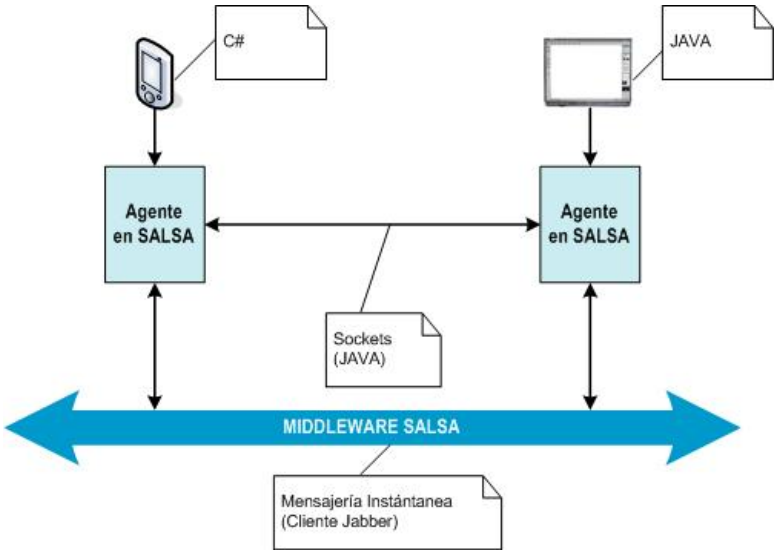


Figura 43. Esquema de las tecnologías utilizadas en el desarrollo del componente de migración.

Las tecnologías utilizadas en el desarrollo del componente pueden agruparse en:

- *Lenguajes de programación:* para las distintas implementaciones se utilizó C# para la versión móvil y, Java para la versión de escritorio.
- *Sistemas operativos:* el componente de migración fue desarrollado bajo las plataformas de Windows XP y Windows CE.
- *Comunicación:* los mecanismos utilizados para la comunicación fueron: mecanismos de SALSA para la versión de escritorio y, se desarrolló mSALSA, un toolkit de SALSA

para trabajar en las PDA bajo la plataforma de C#. Dichos mecanismos utilizan el protocolo de comunicación definido por SALSA. En cuanto a la comunicación necesaria para la transferencia de archivos, se utilizó el mecanismo de Sockets en las dos versiones. Para mayor descripción del mSALSA ver apéndice A.

- *Método de interacción:* se eligió la opción de agregar funcionalidad al menú de opciones del botón derecho del mouse para las distintas plataformas.

V.4.2 Descubrimiento de servicios

Para definir el mecanismo de descubrimiento de servicios se analizaron distintas tecnologías tales como: Jini, LSP, LDAP, entre otros. También se analizó el Directorio de Agentes que proporciona SALSA. Sin embargo, dada la independencia de plataformas que debe proveer el componente, se omitió su uso puesto que dicho directorio se encuentra desarrollado bajo la plataforma de Linux.

Por esta razón, se optó por utilizar la estructura definida en el documento XML. Dicho documento concentra la información de todos aquellos servicios (o dispositivos) que aparecen y desaparecen en el ambiente, los cuales una vez que se encuentran inmersos (conectados) en el ambiente representan los posibles dispositivos destino a los cuales se puede transferir la información.

El esquema que presenta el documento XML es el siguiente:

```
<?xml version="1.0"?>
        <!DOCTYPE device [
           <!ELEMENT device (status, devicetype)>
           <!ATTLIST device name CDATA #IMPLIED>
           <!ELEMENT status (#PCDATA)>
           <!ELEMENT devicetype (#PCDATA)>
        ]>
```

Dicho documento lleva por nombre ***input.xml***, y sirve como archivo de entrada para generar el listado de los dispositivos destino. En base a la estructura del documento, los elementos que contiene son tomados y representados como los posibles dispositivos destino a los cuales puede enviarse la información. Como ejemplo, supóngase la siguiente definición para el documento:

```
<device name="pda@pc-coolab11.cicese.mx">
  <status>O</status>
  <devicetype>pda</devicetype>
</device>
```

De esta forma, se puede identificar que se encuentra disponible en el ambiente el dispositivo *pda@pc-coolab9.cicese.mx*, el cual posee el estado de *Disponible (O = Online)* y es un dispositivo de tipo *pda.*

El descubrimiento de servicios se basa única y exclusivamente en la información que se encuentre contenida dentro del documento. Sin embargo, se contempla la utilización de algunos métodos relacionados con el descubrimiento de servicios tales como: proximidad, frecuencia, localización, entre otros. La implementación de dichos métodos queda fuera del alcance de este trabajo.

A continuación, se presenta el prototipo del componente. Dicho prototipo ilustra la migración de URLs y archivos entre dos dispositivos.

V.4.3 Prototipo

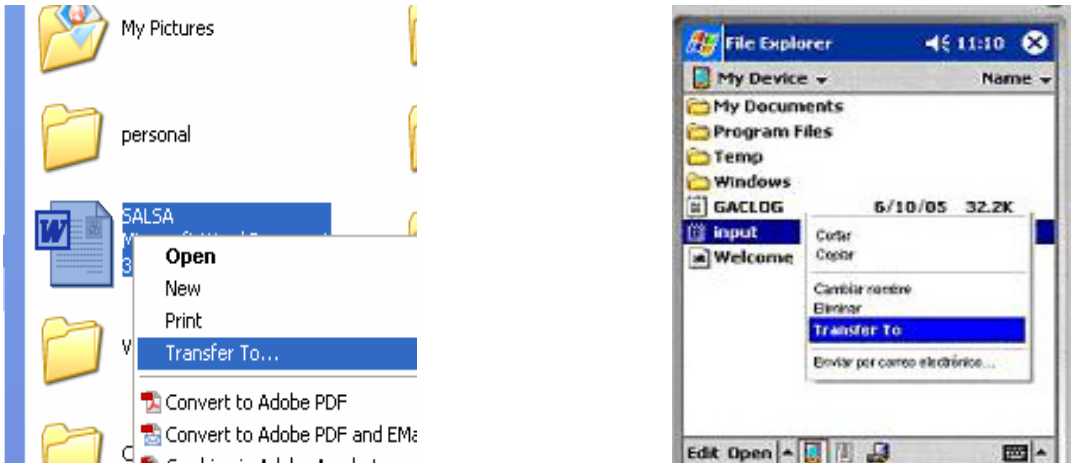
La actual versión del componente de migración provee soporte para la transferencia de: *archivos* y ligas *URL.* El método de interacción elegido para esta versión se encuentra implementado como una alternativa adicional al menú de opciones que proporciona el botón derecho del mouse. El componente consta de dos implementaciones: una para dispositivos de bolsillo (PDA) y otra para computadoras de escritorio (PC, laptop, pantallas públicas, impresoras).

En las siguientes secciones, se describe la transferencia de información (archivos y URLs) en las dos implementaciones del componente.

V.4.3.1 Transferencia de archivos

La transferencia de archivos inicia cuando el usuario selecciona el archivo que desea transferir, y presiona sobre éste con el botón derecho del mouse (en el caso de la PDA,

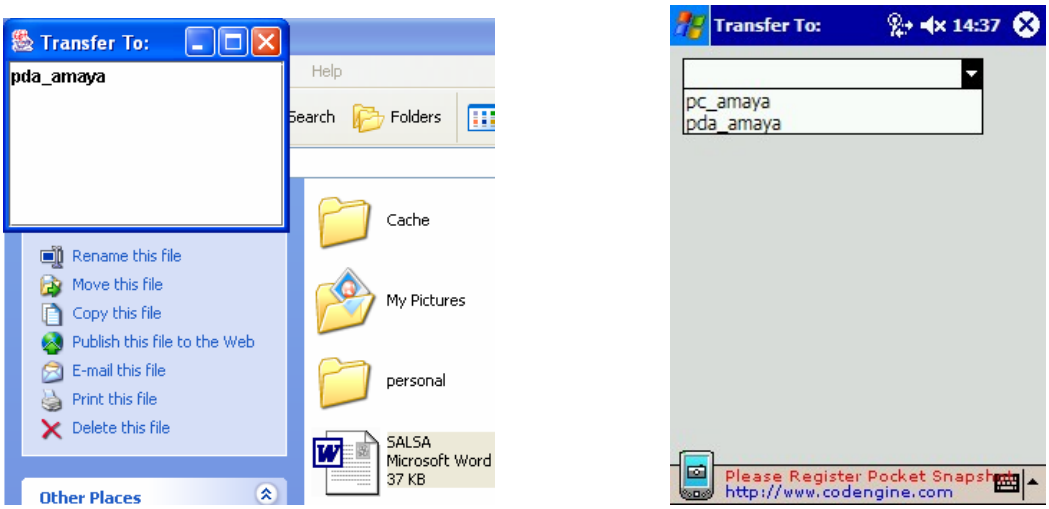
mantiene presionada el cursor hasta que aparezcan las opciones). En la figura 44, se observa el resultado de dicha acción.



44a) Interfaz para dispositivos de escritorio. 44b) Interfaz para la PDA.

Figura 44. Pantalla que ilustra la selección del archivo a transferir.

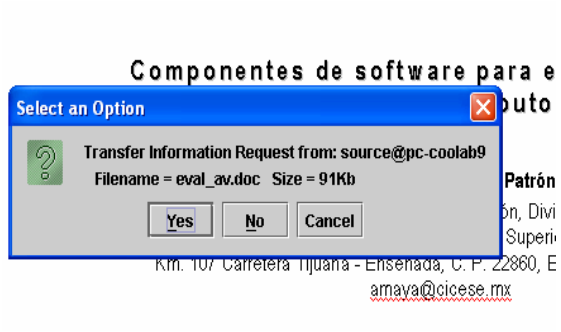
Una vez que el archivo es seleccionado y que aparece el menú de opciones, el usuario debe seleccionar la opción de *Transfer To*. Al seleccionar dicha opción, se despliega una nueva pantalla en la cual aparece el listado de todos aquellos posibles dispositivos destinos a los cuales puede transferirse la información. En la figura 45, puede apreciarse la generación de dicho listado.

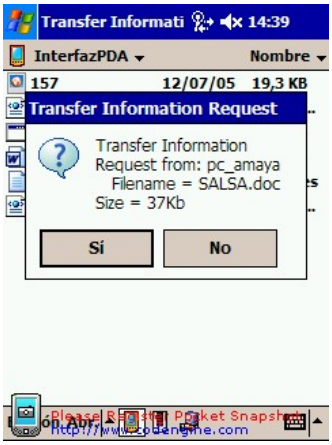


45a) Generación del listado versión de escritorio. 45b) Generación del listado en la PDA.

Figura 45. Pantalla que ilustra la generación del listado de dispositivos destino.

Una vez que se elige el dispositivo destino al cual se desea enviar el archivo, se le solicita a éste si desea aceptar la transferencia. En la figura 46 se ilustra dicha acción.

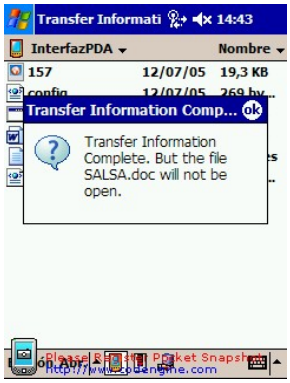




46a) Solicitud de transferencia (versión escritorio). 46b) Solicitud de transferencia (versión PDA).

Figura 46. Pantalla que muestra la solicitud de transferencia del archivo al dispositivo destino.

Si la solicitud de transferencia es aceptada por el dispositivo destino, se da inicio a la migración de la información. Una vez que la información ha sido transferida se notifica al dispositivo origen que se realizó la transferencia (ver figura 47) y, en el dispositivo destino se determina la aplicación asociada al tipo de archivo. En el caso de que el dispositivo destino cuente con la aplicación asociada, se abre la aplicación y se despliega el archivo (ver figura 48).



47a) Notificación de transferencia (versión escritorio). 47b) Notificación de transferencia (PDA).

Figura 47. Pantalla que muestra la notificación de la transferencia.

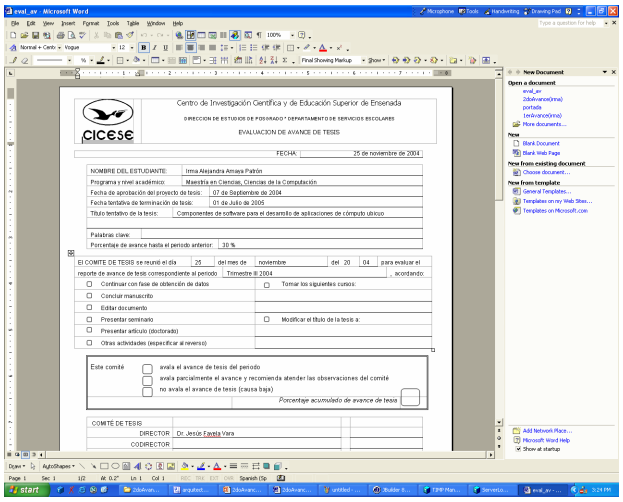


Figura 48. Pantalla que ilustra la apertura del archivo transferido.

V.4.3.2 Transferencia de URLs

El proceso que se sigue para la transferencia de URL es similar al proceso anterior. En este caso, el usuario selecciona la liga (o página web) que desea transferir y da clic con el botón derecho del mouse. Una vez que realiza esta operación, aparece un listado de opciones. Para iniciar con la transferencia del URL debe seleccionarse la opción *Transfer To*. En la figura 49, se muestra la forma en que se realiza la selección de una liga (URL) para transferirse.

Figura 49. Pantalla que ilustra la selección de un URL.

Una vez que la liga ha sido seleccionada y la opción de *Transfer To* ha sido elegida, aparece un listado que contiene la relación de los dispositivos destino a los cuales puede transferirse el URL (ver figura 50).

Figura 50. Pantalla que muestra la relación de dispositivos destino.

Una vez que se selecciona el dispositivo destino, se le envía una notificación en la cual se le solicita la transferencia del URL (ver figura 51).

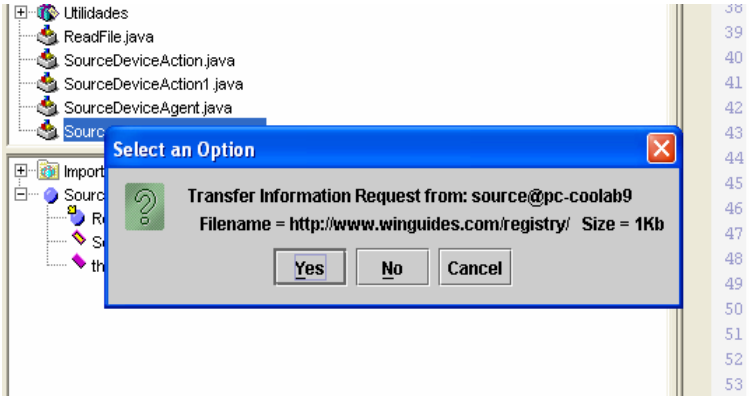


Figura 51. Pantalla que muestra la solicitud de transferencia.

Si se acepta la solicitud de transferencia, se notifica al dispositivo origen que se realizó la migración de información (ver figura 52) y, en el dispositivo destino se determina la aplicación asociada al archivo. En este caso, se abre el navegador con la dirección del URL transferido (ver figura 53).

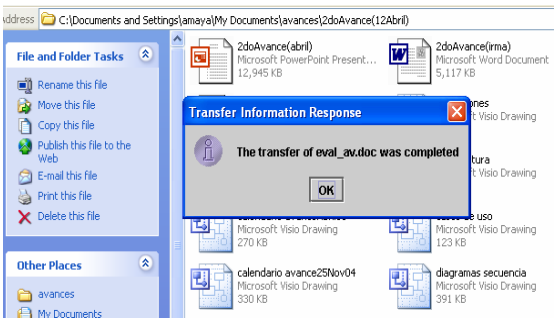


Figura 52. Notificación de aceptación de la transferencia del URL al dispositivo origen.

Figura 53. Apertura del URL en el navegador del dispositivo destino.

En el caso de que la solicitud de transferencia de información sea rechazada, se envía una notificación al dispositivo destino (ver figura 54) y se termina el proceso de migración.

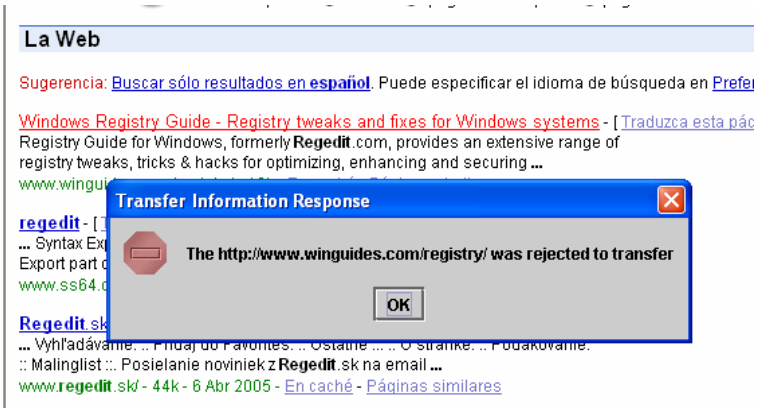


Figura 54. Notificación de rechazo de la transferencia del URL al dispositivo origen.

En la siguiente sección se describen los métodos de interacción implementados para el componente de migración.

V.4.4 Métodos de interacción

El método de interacción presentado en esta etapa comprende la implementación de la modificación a los registros de los sistemas operativos Windows XP y Windows CE para que proporcionen una opción más en el menú de opciones que provee el botón derecho del mouse. Dicha opción aparecerá en el listado que muestra el botón derecho del mouse con el nombre de ***Transfer To***.

Este método representa una forma de interacción con el componente a través de la interfaz definida anteriormente.

Sin embargo, dado que la funcionalidad del componente queda separada completamente de la interfaz es posible implementar otros métodos de interacción. Por esto, se contempla el análisis del método de interacción de arrastre (drag), el cual permitirá completar el escenario del trabajo presentado por [Preciado, 2004].

Este y otros métodos podrán ser implementados por el desarrollador de aplicaciones que busque extender la funcionalidad del componente. Puesto que, a partir de la definición de las entradas del componente, es posible realizar la transferencia de información independientemente del método de interacción que el usuario elija.

En base a esto, para extender o desarrollar un nuevo método de interacción deberá tomarse como base el mensaje que se envía al dispositivo destino solicitando la transferencia de información.

V.5 Resumen

En este capítulo se presentó el diseño del componente de migración. Donde, se definió la arquitectura de dicho componente, para posteriormente definir los elementos del modelo del componente. Continuando con el diseño, se elaboraron diagramas de secuencias, y diagramas de clases. Finalmente, se presentó la implementación del componente, donde se

describieron las tecnologías de desarrollo, así como los mecanismos de descubrimiento de servicios, el prototipo del sistema, y los métodos de interacción utilizados en la implementación del componente.

En el siguiente capítulo se presenta el caso de estudio realizado para la visualización e integración de los componentes de software planteados en este trabajo.

Capítulo VI

Integración de componentes en aplicaciones de cómputo ubicuo

VI.1 Introducción

En este capítulo se presentan algunos escenarios de uso, en los cuales se describen diversas situaciones que pueden resolverse empleando tecnología de cómputo ubicuo. Particularmente, estos escenarios sirven como base para la construcción de soluciones a partir de la integración de componentes de software.

La generación de dichos escenarios tiene como objetivo ilustrar la integración de componentes de software para facilitar y/o agilizar las tareas que se presentan dentro de las situaciones planteadas en estos escenarios. El contexto de los escenarios se sitúa en el sector salud, ya que las características que presentan este tipo de ambientes propician y facilitan el desarrollo de aplicaciones de cómputo ubicuo.

En la segunda parte de este capítulo, se da solución a los escenarios planteados utilizando tecnología de cómputo, particularmente, se utilizaron componentes de software. Los componentes de software utilizados en el desarrollo de estos escenarios son entidades de software independientes desarrolladas por distintos individuos. Con esto, se busca ilustrar como una o más aplicaciones pueden desarrollarse mediante la integración de diversos componentes, donde la forma de interacción entre estos debe definirse claramente para no interferir con la funcionalidad genérica independiente de cada componente.

En las siguientes secciones se describen los escenarios así como la integración de los componentes de software para dar solución a las situaciones planteadas en dichos escenarios.

VI.2 Escenarios

La generación de escenarios de uso tiene como finalidad describir posibles situaciones que podrían presentarse en un ambiente de cómputo ubicuo bajo un contexto hospitalario. A continuación, se presentan los distintos escenarios que fueron identificados:

VI.2.1 Escenario I: *Evaluación colaborativa de expedientes*

Los médicos del hospital se reúnen al inicio del día para evaluar el estado de salud de los pacientes. La reunión se lleva a cabo en la sala de juntas donde se encuentra el pizarrón que contiene una breve historia clínica de los pacientes (ver figura 55). En base a la información que se encuentra en el pizarrón, los médicos analizan los datos y determinan que es necesario evaluar el expediente completo de un paciente. Por esta razón, llaman a una enfermera para solicitarle que les haga entrega del expediente del paciente. Una vez que tienen el expediente, el Dr. González, médico responsable del paciente, toma el expediente y comienza a leer la información; al encontrar datos que requieren otro punto de vista, le solicita al Dr. Méndez que revise y evalúe el expediente. El Dr. Méndez revisa el expediente y determina que es necesario evaluar los rayos X del paciente. Para esto, analizan los rayos x del paciente (ver figura 56) y cada uno de los médicos realiza las observaciones pertinentes a las radiografías, señalando y marcando las áreas o puntos que consideran requieren mayor análisis. Dado que establecen que los rayos X requieren ser revisados por el radiólogo, deciden llamarlo por teléfono, para solicitarle su evaluación. Al establecer la conversación telefónica, el radiólogo les pide que le proporcionen ciertos datos del expediente, así como algunos detalles que puedan observarse en las radiografías, de esta forma darse una idea de la situación en la que se encuentra el paciente. Al tener esta información, comunica su resultado a los médicos quienes continúan con su evaluación. Al finalizar la evaluación, los médicos llaman nuevamente a la enfermera para que se encargue de trasladar el expediente al archivo.

Figura 55. Pizarrón con la información de los pacientes.

Figura 56. Analizando los rayos X del paciente.

VI.2.2 Escenario II: *Monitoreo del estado del paciente*

El sistema de monitoreo detecta una baja en la presión sanguínea del paciente del cuarto 225. Por esta razón, se genera la alerta para informarle al personal encargado de monitorear el estado del paciente que acuda a la sala donde se encuentran los monitores y revise la información y/o el estado de salud del paciente. En atención a la alerta, el Dr. Ramírez decide acudir a la sala de monitoreo y verificar el estado de salud del paciente que sufrió cambio. En base al análisis de la información que realiza de los signos vitales, notifica a las enfermeras las acciones que deben realizar para mejorar el estado de salud del paciente.

VI.3 Solución propuesta: *componentes de software*

En base a la información obtenida de los escenarios anteriores se puede concluir y definir algunos elementos que pueden formar parte de la solución. Estos elementos que se mencionan, son componentes de software que permiten la construcción de aplicaciones de apoyo a estos escenarios en base a la funcionalidad e integración que puede presentarse entre dichos componentes.

A continuación, se describe una variante del escenario I incorporando tecnología de cómputo ubicuo:

> *Los médicos del paciente del cuarto 324 programan una reunión para evaluar su estado de salud. La reunión se lleva a cabo en la sala de juntas donde disponen de distinta tecnología que los auxilia en la toma de decisiones. El Dr. González, médico responsable del paciente, inicia con la evaluación al abrir el expediente del paciente en la pantalla pública para que todos los participantes de la reunión puedan observar la misma información (ver figura 57). Los médicos inician con la evaluación revisando la información que se encuentra en el expediente del paciente; al momento de la revisión, determinan que es necesario evaluar los rayos X del paciente. Para esto, el radiólogo transfiere la información de los últimos rayos X (los cuales aún no se encuentran en el expediente) desde su PDA a la pantalla pública de la sala (ver figura 58). Una vez que transfiere los rayos x,*

decide tomar el control de la reunión, al realizar sus comentarios acerca de las radiografías. Dado que, los demás médicos participantes de la reunión también desean dar su opinión acerca de las radiografías deciden utilizar sus PDAs para interactuar con las radiografías en la pantalla. Para esto, abren la aplicación que les permite visualizar la información de la pantalla y agregan un nuevo apuntador en la pantalla, de esta forma se puede distinguir claramente cuales son los comentarios y observaciones que realiza cada médico (ver figura 59).

Figura 57. Evaluando el expediente del paciente.

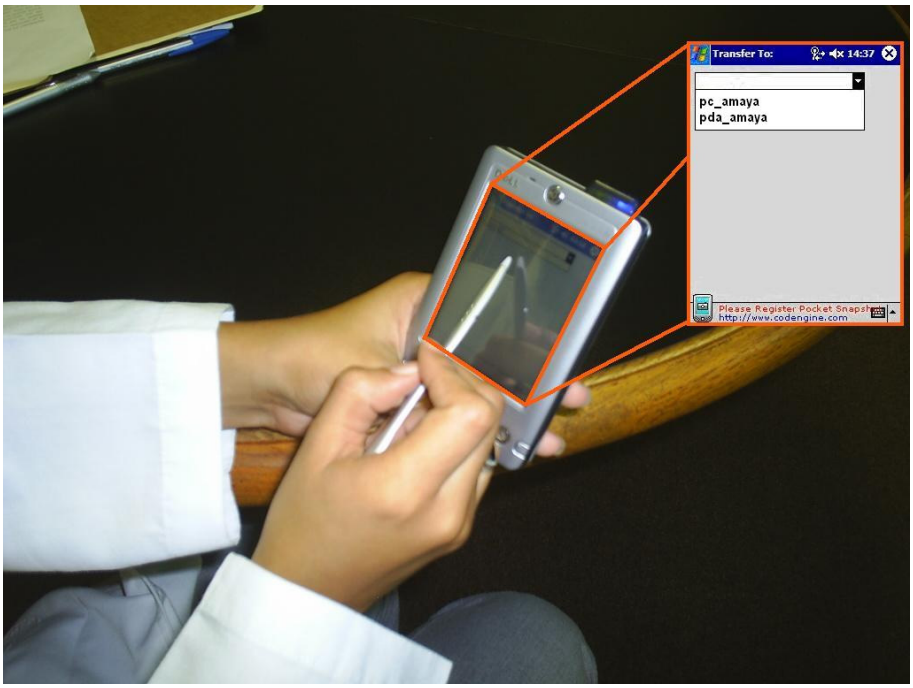


Figura 58. Transfiriendo los rayos X de la PDA a la pantalla pública.

Figura 59. Colaborando a través de la PDA con la pantalla pública.

En base a este escenario, se identifica la secuencia de acciones que se llevan a cabo para lograr el proceso planteado en el escenario. La figura 60, presenta el diagrama de secuencias, para el escenario anterior.

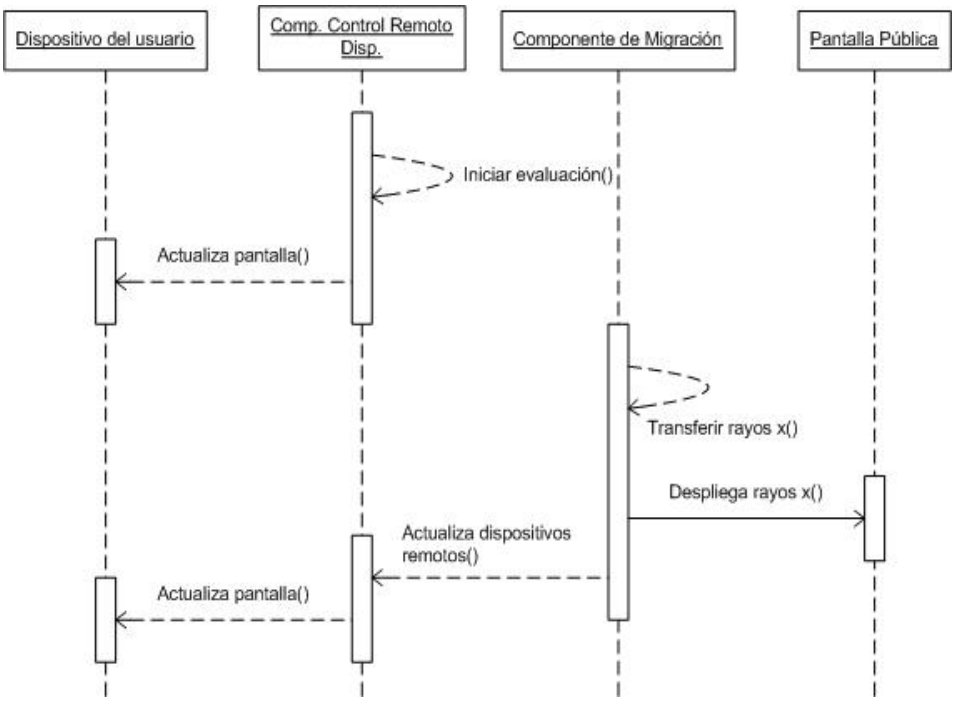


Figura 60. Diagrama de secuencias para el escenario *Evaluación colaborativa de expedientes.*

Los elementos presentados en la figura 60 representan componentes de software cuya interacción permite realizar el flujo de acciones necesarias para facilitar y lograr el objetivo del escenario.

Para el escenario *"Monitoreo del estado del paciente",* se plantea la siguiente solución:

> *Al momento que se detecta una baja en la presión sanguínea del paciente del cuarto 225, el sistema de monitoreo genera la alerta para informarle al Dr. Ramírez que el estado de salud del paciente sufrió un cambio. Dado que, por el momento el Dr. Ramírez se encuentra fuera de la sala de monitoreo, el sistema le envía la imagen del estado de los signos vitales a su dispositivo personal (PDA o teléfono celular). El Dr., al recibir el archivo en su dispositivo decide abrirlo. Al abrir la información, se da cuenta que dada la resolución del dispositivo que lleva*

consigo se le dificulta la lectura de la información. Por esta razón, decide acudir a la sala de monitoreo para ver los resultados. En el camino a la sala, el sistema le detecta que está cerca de una pantalla pública en donde puede abrir el archivo (ver figura 61). El Dr. decide transferir la imagen a la pantalla para ahorrar tiempo. Al momento que llega el Dr. a la pantalla se detecta su presencia y la información que transfirió con anterioridad es desplegada para su evaluación. (ver figura 62) En base al análisis de la información que realiza de los signos vitales, notifica a las enfermeras las acciones que deben realizar para mejorar el estado del paciente.

Figura 61. Notificación de las pantallas públicas más cercanas.

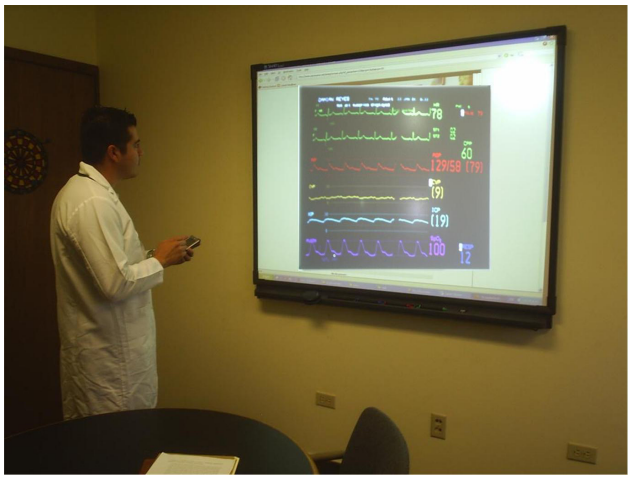


Figura 62. Despliegue de los signos vitales en la pantalla pública.

La figura 63, presenta el diagrama de secuencias, que muestra el flujo de acciones necesarias que se llevan a cabo para lograr el proceso planteado en el escenario.

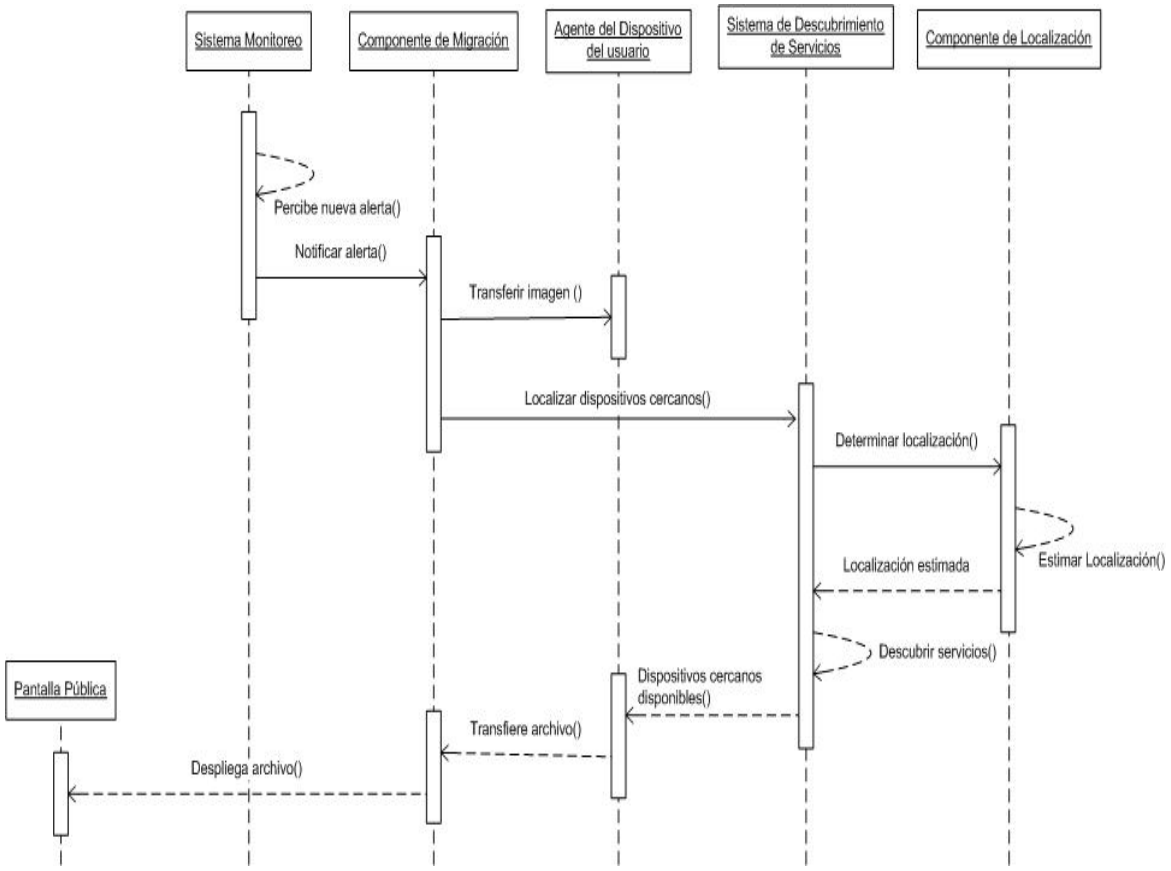


Figura 63. Diagrama de secuencias para el escenario *Monitoreo del estado del paciente.*

Los elementos presentados en la figura 63 representan componentes de software cuya interacción permite realizar el proceso planteado en el escenario anterior.

A continuación, se presenta con mayor detalle la manera en la cual interactúan dichos componentes.

VI.4 Interacción entre componentes

En base a la información recabada en el punto anterior, se identificaron algunos componentes de software tales como: migración, control remoto de dispositivos, y localización. Dichos componentes interactúan entre si para complementar y enriquecer la funcionalidad que estos ofrecen. En la figura 64, se puede observar la forma en que interactúan dichos componentes.

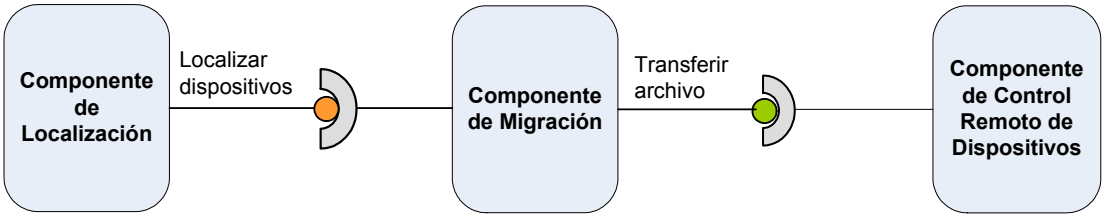


Figura 64. Interacción entre los componentes.

Las interacciones que se presentan entre los componentes solamente podrán ser accedidas a través de las interfaces definidas como: Localizar dispositivos y Transferir archivo. Antes de iniciar la descripción de las interacciones, es necesario definir la arquitectura que tienen en común dichos componentes.

VI.4.1 Definición de la arquitectura

La arquitectura definida para la interacción entre los componentes se presenta en la figura 65. Dicha arquitectura contiene los elementos necesarios para que pueda darse esta interacción entre los componentes.

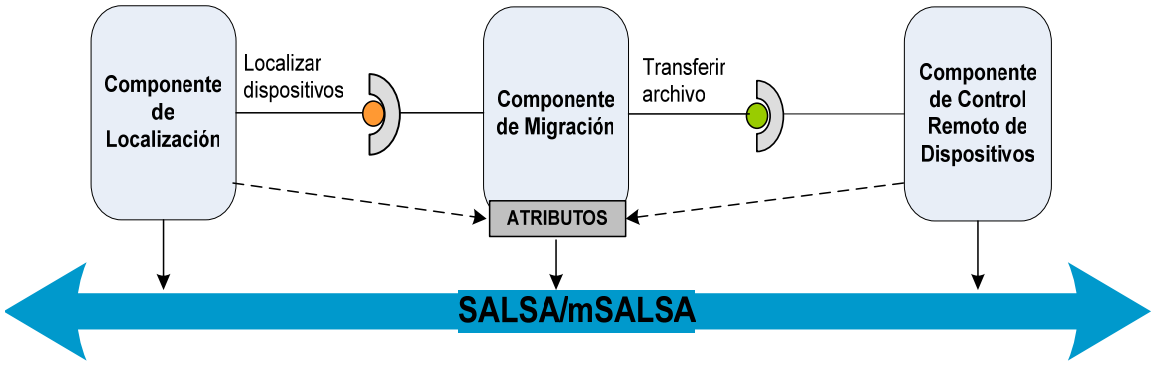


Figura 65. Arquitectura para la Interacción entre los componentes.

Los elementos definidos en la arquitectura son descritos a continuación:

- *Componentes*: los componentes definidos para trabajar bajo esta arquitectura son: localización, migración y control remoto de dispositivos.
- *Middleware SALSA y mSALSA*: los mecanismos y protocolos definidos por el middleware SALSA y mSALSA, para el desarrollo de aplicaciones de escritorio y aplicaciones móviles respectivamente, son el medio a través del cual los componentes pueden comunicarse entre sí y con su ambiente. En base a la información que se obtiene de esta comunicación, los elementos que representan a los componentes pueden efectuar acciones. Por ejemplo, mediante la comunicación establecida para transferir la información, los componentes determinan la apertura de los puertos y servicios necesarios para enviar y/o recibir la información.
- *Atributos*: dada la interacción entre los componentes, los atributos o anotaciones que serán utilizadas para configurar el contexto del componente, son los atributos definidos en el componente de migración. Esto es, los parámetros iniciales de configuración que utilizarán los componentes, serán aquellos valores definidos dentro de la instalación y configuración del componente de migración. Esto con la finalidad, de compartir información común entre los distintos componentes.

A continuación, se define la interacción entre el componente de migración y el componente de localización.

VI.4.2 Interacción migración – localización

Tomando la funcionalidad que provee el componente de localización de [Castro, 2005] y el sistema de descubrimiento de servicios presentado por [Galicia, 2005], se genera un componente de descubrimiento de servicios, que permita localizar y descubrir servicios inmersos en el ambiente mediante una interfaz denominada: *Localizar dispositivos*. El objetivo de esta interfaz es proveer al componente de migración de un listado de dispositivos (servicios) que se encuentren disponibles en su ambiente a los cuales pueda transferir la información. En la figura 66, pueden observarse los atributos y operaciones definidas para esta interfaz.

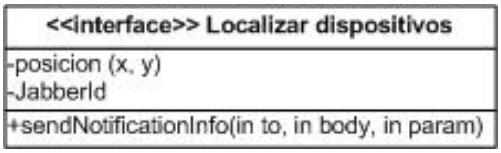


Figura 66. Interfaz *Localizar dispositivos.*

A continuación, se describe cada uno de los elementos que forman parte de los atributos de la interfaz del componente:

- *Posicion (x,y)*: atributo que denota la localización (x, y) del dispositivo en el momento en que se requiere el servicio de localización de dispositivos.
- *JabberClient*: este atributo identifica el cliente jabber del usuario que solicita la localización de dispositivos.

Las operaciones que ofrece la interfaz del componente son las siguientes:

- *sendNotificationInfo*: mediante esta operación se realiza el envío del listado de los dispositivos cercanos. Los parámetros de entrada que recibe son los siguientes:
    - *To*: denota el identificador del usuario a quien se le envía la información.
    - *Body*: es el título que lleva el mensaje que es enviado al usuario. En particular, en esta operación lleva por título: ***Dispositivos cercanos.***

- *Param*: son los parámetros adicionales que se envían en el mensaje para generar el listado de dispositivos disponibles. Los elementos definidos para este parámetro son los siguientes:

  6. ***deviceN***: representa el listado de nombres de los dispositivos cercanos al dispositivo que realizó la solicitud. En este caso, *N* sirve única y exclusivamente para enumerar el listado.

  7. ***positionN***: denota la ubicación del dispositivo.

Como ejemplo, se presenta el siguiente mensaje enviado por la interfaz a través de la operación *sendNotificationInfo*.

```
String mensaje = "<device1>display_pasillo</device1>"
      + "<position1>al final del pasillo</position1>"
      + "<device2>display_conferencias</device2>"
      + "<position2>en la sala de conferencias</position2>"

sendNotificationInfo(to, "Dispositivos cercanos", mensaje);
```

A continuación, se presenta la interacción entre el componente de migración y el componente de control remoto de dispositivos.

VI.4.3 Interacción migración – control remoto de dispositivos

El trabajo realizado por [Markarian, 2005] presenta el componente de control remoto de dispositivos desarrollado sobre SALSA, el cual a su vez, puede interactuar con el componente de migración mediante una interfaz denominada: *Transferir archivo.* El objetivo de esta interfaz es proveer al componente de control remoto de dispositivos de un mecanismo que le permite transferir información dentro de su ambiente. En la figura 67, podemos observar los atributos y operaciones definidas para esta interfaz.

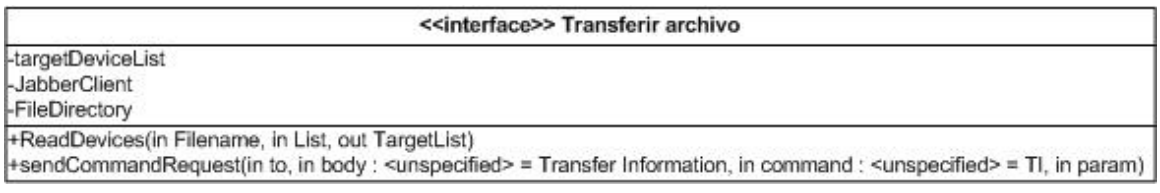


Figura 67. Interfaz *Transferir archivo.*

A continuación, se describe cada uno de los elementos que forman parte de los atributos de la interfaz del componente:

- *targetDeviceList*: atributo que denota la lista de los posibles dispositivos destino con los cuales puede interactuar el usuario para transferir la información.
- *JabberClient*: este atributo denota el cliente jabber a través del cual la interfaz envía la solicitud de transferencia al dispositivo destino.
- *FileDirectory:* este atributo representa el listado de posibles archivos que pueden ser transferidos.

Las operaciones que ofrece la interfaz del componente son las siguientes:

- *ReadDevices*: esta operación permite leer de un archivo XML la lista de los dispositivos destino. La generación de dicha lista es dinámica puesto que, debe considerar únicamente a aquellos dispositivos que se encuentren disponibles en el ambiente al momento de ejecución. Por esta razón, los parámetros de entrada que recibe esta operación son: nombre del archivo (***Filename***), y lista (***List***). Como salida, esta operación regresa los elementos que deben ser agregados a la lista.
- *sendCommandRequest*: mediante esta operación se realiza la solicitud de transferencia al dispositivo destino. Los parámetros de entrada que recibe son los siguientes:
    - *To*: denota el identificador del dispositivo destino a quien se solicita la transferencia de la información.
    - *Body*: es el título que lleva el mensaje que es enviado al dispositivo destino. En particular, esta operación lleva por título: ***Transfer Information.***
    - *Command*: es el comando que se requiere en esta operación. En este caso en particular tiene el valor asignado de: ***TI.***
    - *Params*: son los parámetros adicionales que se envían en el mensaje para identificar la solicitud de transferencia. Los elementos definidos para este parámetro en esta interfaz son los siguientes:

1. ***filename***: representa el nombre del archivo a transferir. Incluye la ruta en donde se encuentra almacenado.
2. ***infotype***: tipo de la información a transferir. Los valores que puede tomar en esta versión del componente son:
    c. ***filetype***, se refiere a la transferencia de un archivo.
    d. ***url***, se refiere a la transferencia de un URL.
3. ***name***: nombre del archivo a transferir (sin la ruta).
4. ***size***: tamaño en Kb, del archivo que se desea transferir. En el caso de la transferencia de URL, el tamaño es igual a 1Kb.
5. ***dtSource***: tipo del dispositivo origen que solicita la transferencia de información. Los tipos de origen pueden ser:
        e. Pda.
        f. Pc.
        g. Laptop.
        h. Display.

Tomemos como ejemplo, el siguiente mensaje enviado por la interfaz a través de la operación *sendCommandRequest*.

```
String enviar = "<filename>C:\\prueba\\prueba.doc</filename>"
      + "<infotype>file</infotype>"
      + "<to>display@pc-coolab9.cicese.mx</to>"
      + "<name>prueba.doc</name>"
      + "<size>36Kb</size>"
      + "<dtSource>pda</dtSource>";

sendCommandRequest(to, "Transfer Information","TI", enviar);
```

VI.5 Resumen

En este capítulo se presentó la integración de componentes de software para dar solución a la construcción de aplicaciones y/o sistemas de cómputo ubicuo. A lo largo de este

capítulo, se plantearon dos escenarios los cuales presentan posibles situaciones que pudieran darse dentro de un ambiente hospitalario. Para dar solución parcial o total a la funcionalidad requerida en dichos escenarios, se propone y describe la utilización de un conjunto de componentes de software, los cuales pueden integrarse para construir aplicaciones complejas. Por último, se definen las distintas interfaces que representan los únicos puntos de acceso a la interacción entre los componentes.

En el siguiente capítulo se presentan las conclusiones, aportaciones y el trabajo futuro de este trabajo de tesis.

Capítulo VII

Conclusiones, aportaciones y trabajo futuro

VII.1 Conclusiones

En este trabajo se realizó un estudio para analizar y determinar la factibilidad del desarrollo de aplicaciones y/o sistemas basados en componentes de software. Dichos componentes, tienen como base la utilización del middleware SALSA.

Con SALSA se permitió la construcción de componentes de software, los cuales pudieron ser representados como agentes para utilizar los mecanismos y protocolos de comunicación definidos por este middleware. De esta forma, se facilitó la construcción de los componentes, permitiendo enfocarnos en la lógica requerida en cada componente.

En este trabajo, los escenarios presentados permiten ejemplificar y distinguir la construcción de ambientes ubicomp basados en la construcción y/o integración de componentes de software que permitan dar solución parcial o total a la funcionalidad definida en dichos escenarios. Algunos de estos, se sitúan en un contexto hospitalario, el cual por las interacciones y situaciones que en este se presentan, representan buenos candidatos para la aplicación de tecnología de cómputo ubicuo.

Estos escenarios, permitieron identificar un conjunto de componentes de software. Para cada uno de estos componentes, se realizó un estudio del estado arte donde se identificó la funcionalidad base que estos deben proveer. Posteriormente, se realizó un análisis comparativo y un estudio de factibilidad para poder determinar el componente a desarrollar.

Debido a las características y la funcionalidad requerida, se decidió iniciar la construcción del componente de migración. Dicho componente, toma como base los mecanismos y protocolos de comunicación definidos en SALSA, para definir y construir agentes de software que permitan en su conjunto llevar a cabo la transferencia de información entre dispositivos. Estos agentes, representan a los diversos dispositivos inmersos en un ambiente ubicomp, y que posee la propiedad de poder transferir y/o recibir información a otro dispositivo.

Algunas de las características que presenta el componente desarrollado son:

- Permite la transferencia de archivos y urls.
- Muestra el listado de los posibles dispositivos destino.
- Permite la configuración y re-configuración de los valores de identificación del usuario.
- La transferencia de información puede realizarse entre dispositivos que pueden ser de naturaleza heterogénea.
- Permite rechazar la transferencia de información.
- Permite desplegar la información transferida.
- Permite almacenar la información trasferida.
- Notifica la transferencia de la información.

Durante el desarrollo de este componente se identificaron algunas interacciones con otros componentes. La primer interacción, se presenta con el componente de control remoto de dispositivos. Esta asociación, permite incrementar la funcionalidad del sistema puesto que la integración de dichos componentes provee mayor funcionalidad al sistema ubicomp. La segunda interacción definida, se presenta con el componente de descubrimiento de servicios. Esta asociación busca la integración del método de descubrimiento de servicios basado en la localización, para poder de esta forma enriquecer el contexto de la información necesaria para permitir la transferencia de información. Aunado a estas interacciones, existen otros componentes tales como: autentificación, agenda y

colaboración, los cuales pueden integrarse conjuntamente para construir un ambiente ubicomp complejo y funcional.

En este trabajo se generó y ejemplificó una guía para el desarrollo basado en componentes. Dicha guía proporciona una serie de pasos y elementos que debe cumplir una aplicación y/o sistema para considerarse un componente de software. Esta guía permite a los desarrolladores de software analizar la factibilidad de implementar sus aplicaciones y/o sistemas basados en componentes.

Con esto, se concluye que identificar un conjunto de componentes, desarrollarlos, ampliar el alcance de SALSA y desarrollar aplicaciones en base a los componentes desarrollados permiten cumplir con el objetivo principal de este trabajo, el cual es desarrollar un sistema de componentes que incrementen el alcance de SALSA y permitan el fácil desarrollo de aplicaciones para cómputo ubicuo.

VII.2 Aportaciones

Dentro de las aportaciones de este trabajo de investigación se pueden mencionar:

- Se desarrolló un componente de software que permite la transferencia de información en ambientes ubicuos. El componente permite la migración de información (archivos y urls) entre diversos dispositivos que pueden ser heterogéneos.
- Se provee un mayor nivel de abstracción en el desarrollo de aplicaciones, puesto que el desarrollo basado en componentes nos permite reutilizar y extender la funcionalidad de los mismos, sin conocer la lógica de su implementación.
- Los escenarios identificados nos permiten identificar diversas situaciones que se presentan recurrentemente en ambientes de cómputo ubicuo.
- Utilización del componente de migración en el proyecto *Diseminación de información a través de pantallas públicas conscientes del contexto* de la Facultad de Ingeniería de la Universidad Autónoma de Baja California (UABC).

- Se desarrolló una versión de SALSA para dispositivos móviles denominada mSALSA, la cual permite el desarrollo de aplicaciones móviles en ambientes ubicomp.
- Los mecanismos de transferencia de información implementados en el componente de migración sirvieron como base para la implementación del componente de control remoto de dispositivos.
- Se incrementó el alcance de SALSA al crear componentes de software que pueden integrarse naturalmente en un ambiente de cómputo ubicuo.

VII.3 Trabajo Futuro

El alcance definido para este trabajo está terminado. Sin embargo, durante su desarrollo surgen ciertos aspectos que pueden considerarse como trabajo futuro. A continuación, se describen algunos de estos aspectos:

- Actualmente el componente de migración, realiza una simulación del descubrimiento de servicios, puesto que se basa únicamente en la disponibilidad del dispositivo. Por esto, se considera integrar distintos métodos de descubrimiento de servicios que se apeguen mejor a un ambiente ubicomp.
- Evaluar la integración y adaptación del componente de migración dentro del proyecto presentado por la UABC. Este proyecto, plantea la utilización de pantallas públicas, las cuales contienen información que será desplegada conforme a ciertos parámetros (preferencias, fechas, lugar, etc.) del usuario. El uso del componente de migración se plantea para transferir notas desde diversos dispositivos hacia la pantalla pública y viceversa. De esta forma, se pretende demostrar que el componente de migración puede ser utilizado tanto para transferir como para adaptar la información que es publicada en pantallas publicas.
- Probar la integración del componente de migración en distintos contextos. Esto con la finalidad de demostrar los beneficios y ventajas que proporciona el desarrollo basado en componentes.

Referencias

Abowd, G. D., 1999. "Software Engineering Issues for Ubiquitous Computing". En: Proceedings of the 21st International Conference on Software Engineering (ICSE'99). Los Angeles, CA. IEEE Computer Society Press. 75-84 p.

Banavar, G. y Bernstein, A., 2002. "Software infrastructure and design challenges for ubiquitous computing applications". Communications of the ACM. 45(12): 92-96 p.

Bardram, J. E., 2004. "Applications of context-aware computing in hospital work: examples and design principles". En: Proceedings of the 2004 ACM Symposium on Applied Computing (ACM SAC). ACM Press. 1574-1579 p.

Bardram, J. E., 2005. "Activity-Based Computing: Support for Mobility and Collaboration in Ubiquitous Computing". Personal and Ubiquitous Computing. 9(5): 312-322 p.

Bardram, J. E., 2003. "Hospital of the future – ubiquitous computing support for medical work in hospitals". Leído el 17 de Septiembre de 2004. http://www.daimi.au.dk/~bardram/docs/future.hospital.ubihealth.pdf

Bardram, J. E., Kjær, R. E. y Pedersen, M. Ø., 2003. "Context-aware user authentication-supporting proximity-based login in pervasive computing". En: Dey, A. K., Schmidt, A. y McCarthy, J. F. (eds.). UbiComp 2003: Ubiquitous Computing: 5th International Conference, Lecture Notes in Computer Science. Seattle, Washington, United States. Springer-Verlag. 2864: 107-123 p.

Barr, P., 1999. "Component-Based Architecture Development". Leído el 09 de Septiembre de 2004. http://www.stc-online.org/cd-rom/1999/slides/SofComp8.pdf

Baudish, P., Cutrell, E., Robbins, D., Czerwinski, M., Tandler, P., Bederson, B. y Zierlinger, A., 2003. "Drag-and-Pop and Drag-and-Pick: techniques for accessing remote screen content on touch-and pen operated systems". En: Proceedings of Interact 2003. Zurich Switzerland. 57-64 p.

Bellifemine, F., Caire, G., Poggi, A. y Rimassa, G., 2003. "JADE: A White Paper". En: In search of innovation. 3(3): 6-19 p.

Bellotti, V., Dalal, B., Good, N., Flynn, P., Bobrow, D. G. y Ducheneaut, N., 2004 "What a to-do: studies of task management towards the design of a personal task list manager". En: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. Vienna, Austria. ACM Press. 735-742 p.

Boertien, N., Jonkers, H., Slagter, T. y Steen, M., 2001. "Component-based rapid service development: Methodologies, modeling and use of components in the context of the RSD". Reporte Técnico. Telematica Instituut, The Netherlands. 72 pp.

Cai, X., Lyu, M. R., Wong, K. y Ko, R., 2000. "Component-based software engineering: technologies, development frameworks, and quality assurance schemes". En: Proceedings of the Seventh Asia-Pacific Software Engineering Conference. IEEE Computer Society. Washington. DC, USA. 372-381 p.

Castro, L., 2005. "Estimación de localización de dispositivos móviles en redes de radio frecuencia utilizando series de tiempo". Centro de Investigación Científica y de Educación Superior de Ensenada. Tesis de Maestría en Ciencias en Ciencias de la Computación. 95 pp.

Cerelli, B., 2005. "Registry Tips". Leído el 10 de Marzo de 2005. http://www.onecomputerguy.com/registry_tips.htm

Coulson, G., Blair, G., Hutchison, D., Joolia, A., Lee, K., Ueyama, J., Gomes, A. y Ye, Y., 2003. "NETKIT: A Software Component-Based Approach to Programmable Networking". ACM SIGCOMM Computer Communications Review (CCR). 33(5): 55-66 p.

Crnkovic, I., 2001. "Component-based software engineering – new challenges in software development". Software Focus. John Wiley & Sons. 2(4): 127-133 p.

Crnkovic, I., Hnich, B., Jhonsson, T. y Kiziltan, Z., 2002. "Specification, implementation, and deployment of components". Communications of the ACM. 45(10): 35-40 p.

C# Help., 2005. "C# / JAVA Socket Programming". Leído el 20 de Marzo de 2005. http://www.csharphelp.com/archives2/archive434.html

Dey, A. K. y Abowd, G. D., 2000. "CybreMinder: A context-aware system for supporting reminders". En: Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing. Bristol, UK. Springer-Verlag. 172-186 p.

Dey, A. K., Hamid, R., Beckmann, C., Li, I. y Hsu, D., 2004. "a CAPpella: programming by demonstration of context-aware applications". En: Proceedings of the 2004 conference on Human factors in computing systems. Vienna, Austria. 6(1): 33-40 p.

Favela, J., Rodríguez, M., Preciado, A. y González, V. M., 2004. "Integrating context-aware public displays into a mobile hospital information system". IEEE Transactions on Information Technology in Biomedicine. 8(3): 279-286 p.

Fitzgibbon, J., 2003. "Customizing Right-Click Menu Options in Windows". Leído el 15 de Marzo de 2005. http://www.jfitz.com/tips/rclick_custom.html

Fitzgibbon, J., 2002. "Search from IE's right-click menu". Leído el 15 de Marzo de 2005. http://www.jfitz.com/tips/search.htm

Galicia, L., 2005. “Descubrimiento de servicios basados en proximidad física”. Centro de Investigación Científica y de Educación Superior de Ensenada. Tesis de Maestría en Ciencias en Ciencias de la Computación. 161 pp.

Gilleade, K., Sheridan, J. y Allanson, J., 2003. “Liquid: Designing a Universal Sensor Interface for Ubiquitous Computing”. Reporte Técnico, comp-001-2003. Lancaster University. Lancaster, UK. 8 pp.

Greenberg, S., 2004. “Collaborative Physical User Interfaces”. Reporte Técnico, 2004-740-05. Department of Computer Science. University of Calgary. Calgary, Alberta, Canada. 19 pp.

Greenberg, S., Boyle, M. y Laberge, J., 1999. “PDAs and shared public displays: making personal information public, and public information personal”. Personal Technologies. 3(1): 54-64 p.

Griss, M. L., 2003. “Software Engineering with Java Agent Components”. Leído el 10 de septiembre de 2004. http://www.soe.ucsc.edu/~griss

Griss, M. L., 2001. “Software Agents as Next Generation Software Components”. En: Heineman, G. T. y Councill, W. “Component-Based Software Engineering: Putting the Pieces Together”. Addison-Wesley, Boston, EUA. 641-657 p.

Griss, M. L. y Pour, G., 2001. “Accelerating development with agent components”. IEEE Computer. 34(5): 37-43 p.

Hooper, A., 2003. “NDNM3: An ActiveX Control for embedding NetMeeting in your VB/ATL apps”. Leído el 30 de septiembre de 2004. http://www.hoppersoft.com/NDNM3

IEEE Std 830-1998, 1998. “IEEE recommended practice for software requirements specifications”. Leído el 30 de noviembre de 2004. http://kybele.escet.urjc.es/Documentos/ISI/IEEE-STD-830-1998.pdf

Johanson, B., Fox, A. y Winograd, T., 2002. “The interactive workspaces project: experiences with ubiquitous computing rooms”. IEEE Pervasive Computing. 1(2): 67-74 p.

Johanson, B., Ponnekanti, S., Sengupta, C. y Fox, A., 2001. “Multibrowsing: moving web content across multiple displays”. En: Proceedings of the 3rd international conference on Ubiquitous Computing. Atlanta, Georgia, EUA. Springer-Verlag. 2201: 346-353 p.

Kaowthumrong, K., Lebsack, J. y Han, R., 2002. “Automated Selection of the Active Device”. En: Workshop at UbiComp’02: Supporting Spontaneous Interaction in Ubiquitous Computing Settings, Gteborg, Sweden. 1-6 p.

Kindberg, T. y Fox, A., 2002. "System software for ubiquitous computing". IEEE Pervasive Computing. 1(1): 70-81 p.

Konomi, S., Müller-Tomfelde, C., y Streitz, N., 1999. "Passage: physical transportation of digital information in cooperative buildings". En: Proceedings of the 2nd International Workshop on Cooperative Buildings (CoBuild'99). Heidelberg, Germany. Springer-Verlag. 1670: 45-54 p.

Landry, B. M., Nair, R. y Tungare, M., 2004. "TaskMinder: an intelligent adaptive to-do list". Leído el 05 de Octubre de 2004. http://www.manastungare.com/publications/taskminder/TaskMinder.pdf

Long, S., Kooper, R., Abowd, G. D. y Atkeson, C. G., 1996. "Rapid prototyping of mobile context-aware applications: the Cyberguide case study". En: Proceedings of the 2nd Annual International Conference on Mobile Computing and Networking. Rye, New York, EUA. 97-107 p.

Markarian, A., 2005. "Ubiquitous computing support for collaboration un hospital work". University of Bristol, United Kingdom. Centre of Scientific Research and Higher Education of Ensenada, Mexico. Telecommunication Department, INP Grenoble, France. Reporte Técnico. 52 pp.

McInnis, K., 2000. "Component-based Development: The concepts, technology and methodology". Castek Software Factory Inc. Leído el 1 de Septiembre de 2004. http://www.cbd-hq.com/PDFs/cbdhq_000901km_cbd_con_tech_method.pdf

McInnis, K., 1999. "Component-based Design and Reuse". Castek Software Inc. Leído el 10 de septiembre de 2004. http://www.cbd-hq.com/PDFs/cbdhq_990715km_CBDreuse.pdf

Paradiso, J. A. y Feldmeier, M., 2001. "A Compact, Wireless, Self-Powered Pushbutton Controller Source". En: Proceedings of the 3rd international conference on Ubiquitous Computing. Atlanta, Georgia, EUA. Springer-Verlag. 2201: 299-304 p.

PCStats, 2005. "Beginners Guides: 104 Performance Tips for Windows XP". Leído el 20 de Marzo de 2005. http://www.pcstats.com/

Preciado, A., 2004. "Interacción con pantalla públicas en ambientes médicos ubicuos". Centro de Investigación Científica y educación Superior de Ensenada. Tesis de Maestría en Ciencias en Ciencias de la Computación. 181 pp.

Reid, A., Flatt, M., Stoller, L., Lepreau, J. y Eide, E., 2000. "Knit: Component Composition for Systems Software". En: Proceedings of 4th Symposium on Operating Systems Design and Implementation (OSDI 2000). Usenix Association. 347-360 p.

Rekimoto, J., 1997. "Pick-and-drop: a direct manipulation technique for multiple computer environments". En: Proceedings of the 10th annual ACM symposium on User interface software and technology. Banff, Alberta, Canada. ACM Press. 31-39 p.

Ritchie, M., 2002. "Pre & post processing for service based context-awareness". Technical report Equator-02-023. Department of Computing Science. University of Glasgow. United Kingdom. 4 pp.

Rodríguez, M. D., Andrade, A. G., y González, M., en proceso. "Diseminación de Información a través de Pantallas Publicas Concientes del Contexto". Por aparecer en el 2do. Congreso Latinoamericano de Interacción Humano-Computadora (CLIHC 2005). 2 pp.

Rodríguez, M., Favela, J., Preciado, A., y Vizcaino, A., 2005. "Agent-based ambient intelligence for healthcare". AI Communications. IOS Press. 1-16 pp.

Rodríguez, M., Favela, J. y Preciado, A., 2004. "An agent middleware for supporting ambient intelligence for healthcare". En: Second workshop on agents applied in Health Care, Held in conjunction with the 16th European Conference on Artificial Intelligence (ECAI 2004). Valencia, Spain. 8 pp.

Roseman, M. y Greenberg, S., 1997. "Building Groupware with GroupKit". En: M. Harrison. Tcl/Tk Tools. O'Reilly Press. 535-564 p.

Satyanarayanan, M., 2002. "A Catalyst for Mobile and Ubiquitous Computing". IEEE Pervasive Computing. 1(1): 2-6 p.

Spreitzer, M. y Theimer, M., 1993. "Providing location information in a ubiquitous computing environment". En: Proceedings of the 14th ACM symposium on Operating systems principles. Asheville, North Carolina, EUA. 270-283 p.

Streitz, N. A., Geißler, J., Holmer, T., Konomi, S., Müller-Tomfelde, C. Reischl, W., Rexroth, P., Seit, P. y Steinmetz, R., 1999. "i-LAND: an interactive landscape for creativity and innovation". En: Conference on Human factors in computing systems. Proceedings of the SIGCHI on Human factors in computing systems: the CHI is the limit. Pittsburgh, Pennsylvania, United States. 120-127 p.

Supriya, P., 2002. "Introduction to component-based software development (CBSD)". Center for Development of Advanced Computing. Pune University Campus, Ganesh Khind, India. Component Software Module. Lecture Notes. 10 pp.

Szyperski, C., Gruntz, D., y Murer, S., 2002. "Component Software – Beyond Object – Oriented Programming". Addisson-Wesley/ACM Press. 2nd Edition. Great Britain. 608 pp.

Tandler, P., 2003. "The BEACH application model and software framework for synchronous collaboration in ubiquitous computing environments". En: Journal of Systems & Software, (special issue on Ubiquitous Computing edited by J.J. Barton, R. Cerqueira and M. Fontoura). 69(3): 267-296 p.

Tentori, M., 2005. "Cómputo consciente de la privacidad en ambientes médicos ubicuos", Centro de investigación Científica y de Educación Superior de Ensenada. Tesis de Maestría en Ciencias en Ciencias de la Computación. 170 pp.

Villar, N., Kortuem, G., Van-Laerhoven, K. y Schmidt, A., en proceso. "The Pendle: A Personal Mediator for Mixed Initiative Environments". Leído el 15 de Agosto de 2005. http://ubicomp.lancs.ac.uk/~villar/PDF/pendle-ie05.pdf

Völter, M., 2002. "Components, Remoting Middleware and Webservices – and how it all fits together". Presentado en: OOP 2003. Munich, Germany. 51 pp.

Wallbank, N., 2002. "A requirements analysis of infrastructure for ubiquitous computing environments". Technical report Equator-02-042. Computing Department. Lancaster University. United Kingdom. 27 pp.

Weiser, M., 1996. "Ubiquitous Computing". Leído el 5 de septiembre de 2004. http://www.ubiq.com/hypertext/weiser/UbiHome.html

Apéndice A.

API de mSALSA

En este apéndice se presenta el API desarrollado para mSALSA. La información contenida en dicho apéndice describe las distintas clases que proporciona mSALSA. mSALSA es la versión de SALSA que permite el desarrollo de aplicaciones móviles, siguiendo la misma teoría y lógica implementada en SALSA, la cual es habilitar la construcción de agentes de software para el desarrollo de aplicaciones ubicomp. De esta forma, mSALSA permite el desarrollo de aplicaciones móviles ubicomp basados en agentes.

La implementación de mSALSA se realizó utilizando el lenguaje de programación C#, el cual facilitó el desarrollo del API, al proporcionar librerías de clases que permitieron efectuar la comunicación entre agentes que representan en este a caso a dispositivos móviles. Particularmente, el ambiente de desarrollo de mSALSA fue probado bajo la plataforma de Windows CE, específicamente en un dispositivo de tipo Pocket PC.

A continuación, describimos las que conforman el mSALSA.

A.1 Clases

En los siguientes puntos describimos cada una de las clases que en su conjunto representan el middleware mSALSA.

A.1.1 Clase *Acting*

Componente de acción del agente.

Declaración: *public class Acting : Object*

Constructores

Constructor	Declaración	Descripción	Parámetros
.ctor	public Acting()	Recibe el cliente jabber para utilizarlo en la clase *Action.*	jClient

Métodos

Método	Declaración	Descripción	Parámetros
act	public void act()	Ejecuta una acción	action
Equals	public bool Equals()	Determina si el objeto específico es igual al actual.	obj
Finalize	protected void Finalize()	Permite liberar recursos y realizar otras operaciones de la limpieza antes de que sea reclamado por la colección de la basura	
GetHashCode	public int GetHashCode()	Sirve como una función hash para un tipo particular, conveniente para el uso de algoritmos hash y estructuras de datos tales como tablas hash	
GetType	public System.Type GetType()	Obtiene el tipo de la instancia actual.	
MemberwiseClone	protected object MemberwiseClone()	Crea una copia bajo el objeto actual.	
ToString	public string ToString()	Regresa una cadena de caracteres que representan al objeto actual.	

A.1.2 Clase *Action*

Acción del agente. El desarrollador especifica en esta clase la accion del o el conjunto de acciones del agente.

Declaración: *public class Action : Object*

Constructores

Constructor	Declaración	Descripción	Parámetros
.ctor	public Action()	Constructor de la clase *Action.*	

Métodos

Método	Declaración	Descripción	Parámetros
Equals	public bool Equals()	Determina si el objeto específico es igual al actual.	obj
execute	public void execute()	Este método es implementado por el desarrollador para especificar la acción de un agente.	
Finalize	protected void Finalize()	Permite liberar recursos y realizar otras operaciones de la limpieza antes de que sea reclamado por la colección de basura	
GetHashCode	public int GetHashCode()	Sirve como una función hash para un tipo particular, conveniente para el uso de algoritmos hash y estructuras de datos tales como tablas hash	
GetType	public System.Type GetType()	Obtiene el tipo de la instancia actual.	
MemberwiseClone	protected object MemberwiseClone()	Crea una copia bajo el objeto actual.	
sendCommandRequest	public void sendCommandRequest()	Envía un mensaje para solicitar la ejecución de un comando o acción a otro agente.	to, body, command, param
sendMessage	public void sendMessage()	Envía un mensaje a otro agente.	to, body, type, param
sendMessage	public void sendMessage()	Envía un simple mensaje a otro agente. ***Uso:*** Envía un mensaje xml simple	from, to, body
sendNotificationInfo	public void sendNotificationInfo()	Envía una notificación.	to, body, type, params
sendPresence	public void sendPresence()	Envía un mensaje de presencia.	status, nickUser, area, location, typeOfAgent
sendRequest	public void sendRequest()	Envía un mensaje para solicitar información a otro agente.	to, body, type, params
sendResponse	public void sendResponse()	Envía un mensaje que responde a una solicitud de información de otro agente.	To, body, type params
setProxyBroker	public void setProxyBroker()	Este método crea una instancia del cliente jabber para la clase *Action*.	jClient
ToString	public string ToString()	Regresa una cadena de caracteres que representan al objeto actual.	

A.1.3 Clase *Agent*

Clase del Agente mSalsa.

Declaración: *public class Agent : Object*

Campos

Campos	Declaración	Descripción	Parámetros
passivePerception	public mSalsa.PassivePerception passivePerception	Instancia de la clase *PassivePerception.*	

Constructores

Constructor	Declaración	Descripción	Parámetros
.ctor	public Agent()	Crea una instancia del agente.	

Métodos

Método	Declaración	Descripción	Parámetros
activate	protected void activate()	Crea y activa un agente que se comunica a través de mensajería instantánea con otros agentes.	rsn, jClient
Equals	public bool Equals()	Determina si el objeto específico es igual al actual.	obj
Finalize	protected void Finalize()	Permite liberar recursos y realizar otras operaciones de la limpieza antes de que sea reclamado por la colección de la basura	
GetHashCode	public int GetHashCode()	Sirve como una función hash para un tipo particular, conveniente para el uso de algoritmos hash y estructuras de datos tales como tablas hash	
GetType	public System.Type GetType()	Obtiene el tipo de la instancia actual.	
MemberwiseClone	protected object MemberwiseClone()	Crea una copia bajo el objeto actual.	
ToString	public string ToString()	Regresa una cadena de caracteres que representan al objeto actual.	

A.1.4 Clase *Event*

Clase del Evento.

Declaración: *public class Event : Object*

Propiedades

Propiedad	Declaración	Descripción
input	public mSalsa.Input input { get; set; }	Obtiene el objeto *input.*
type	public mSalsa.salsa event type type { get; set; }	Obtiene el tipo de evento.
xml	public string xml { get; set; }	Obtiene el mensaje XML.

Constructores

Constructor	Declaración	Descripción	Parámetros
.ctor	public Event()	Crea un evento que el mensaje xml percibido.	xml, obj

Métodos

Método	Declaración	Descripción	Parámetros
Equals	public bool Equals()	Determina si el objeto específico es igual al actual.	obj
Finalize	protected void Finalize()	Permite liberar recursos y realizar otras operaciones de la limpieza antes de que sea reclamado por la colección de la basura	
GetHashCode	public int GetHashCode()	Sirve como una función hash para un tipo particular, conveniente para el uso de algoritmos hash y estructuras de datos tales como tablas hash	
GetType	public System.Type GetType()	Obtiene el tipo de la instancia actual.	
MemberwiseClone	protected object MemberwiseClone()	Crea una copia bajo el objeto actual.	
ToString	public string ToString()	Regresa una cadena de caracteres que representan al objeto actual.	

A.1.5 Clase *Input*

Clase *Input.*

Declaración: *public class Input : Object*

Propiedades

Propiedad	Declaración	Descripción
data	public object data { get; set; }	Valor de los datos.

Constructores

Constructor	Declaración	Descripción	Parámetros
.ctor	public Input()	Recibe un objeto como entrada.	data

Métodos

Método	Declaración	Descripción	Parámetros
Equals	public bool Equals()	Determina si el objeto específico es igual al actual.	obj
Finalize	protected void Finalize()	Permite liberar recursos y realizar otras operaciones de la limpieza antes de que sea reclamado por la colección de la basura	
GetHashCode	public int GetHashCode()	Sirve como una función hash para un tipo particular, conveniente para el uso de algoritmos hash y estructuras de datos tales como tablas hash	
GetType	public System.Type GetType()	Obtiene el tipo de la instancia actual.	
MemberwiseClone	protected object MemberwiseClone()	Crea una copia bajo el objeto actual.	
ToString	public string ToString()	Regresa una cadena de caracteres que representan al objeto actual.	

A.1.6 Clase *JabberClient*

Clase para manejar el cliente jabber agsXMPP.

Declaración: *public class JabberClient : XmppClientConnection*

Propiedades

Propiedad	Declaración	Descripción
Authenticated	public bool Authenticated { get; }	
AutoAgents	public bool AutoAgents { get; set; }	
AutoRoster	public bool AutoRoster { get; set; }	
Binded	public bool Binded { get; }	
IqGrabber	public agsXMPP.IqGrabber IqGrabber { get; }	
KeepAlive	public bool KeepAlive { get; set; }	
KeepAliveInterval	public int KeepAliveInterval { get; set; }	
MesagageGrabber	public agsXMPP.MessageGrabber MesagageGrabber { get; }	
MyJID	public agsXMPP.Jid MyJID { get; }	
Password	public string Password { get; set; }	
Port	public int Port { get; set; }	
Priority	public int Priority { get; set; }	
Resource	public string Resource { get; set; }	
Server	public string Server { get; set; }	
Show	public agsXMPP.protocol.ShowType Show { get; set; }	
Status	public string Status { get; set; }	
StreamId	public string StreamId { get; }	
StreamParser	public agsXMPP.StreamParser StreamParser { get; }	
StreamVersion	public string StreamVersion { get; set; }	
Username	public string Username { get; set; }	
UseSSL	public bool UseSSL { get; }	

Constructores

Constructor	Declaración	Descripción	Parámetros
.ctor	public JabberClient()	Conectarse al servidor jabber	server, port, user, pass

Métodos

Método	Declaración	Descripción	Parámetros
Close	public void Close()	Cierra la instancia del cliente jabber.	
Equals	public bool Equals()	Determina si el objeto específico es igual al actual.	obj
Finalize	protected void Finalize()	Permite liberar recursos y realizar otras operaciones de la limpieza antes de que sea reclamado por la colección de la basura	
GetHashCode	public int GetHashCode()	Sirve como una función hash para un tipo particular, conveniente para el uso de algoritmos hash y estructuras de datos tales como tablas hash	
GetType	public System.Type GetType()	Obtiene el tipo de la instancia actual.	
MemberwiseClone	protected object MemberwiseClone()	Crea una copia bajo el objeto actual.	
Open	public void Open()	Abre una conexión en el servidor jabber.	
Open	public void Open()	Abre una conexión autentificada.	username, password
Open	public void Open()	Abre una conexión autentificada especificando el recurso.	username, password, resource
Open	public void Open()	Abre una conexión autentificada especificando el recurso y la prioridad	username, password, resource, prioridad
Open	public void Open()	Abre una conexión autentificada especificando la prioridad	username, password, prioridad
RequestAgents	public void RequestAgents()		
RequestRoster	public void RequestRoster()		
Send	public void Send()	Envía el mensaje xml.	xml
Send	public void Send()	Envía el mensaje e.	e
SendMyPresence	public void SendMyPresence()		
SendOpenStream	public void SendOpenStream()		startParser
setReasoning	public void setReasoning()	Asigna el componente de razonamiento al cliente jabber.	rsn
ToString	public string ToString()	Regresa una cadena de caracteres que representan al objeto actual.	

A.1.7 Clase *Parser*

Sencillo parser xml.

Declaración: *public class Parser : Object*

Constructores

Constructor	Declaración	Descripción	Parámetros
.ctor	public Parser()	Crea una instancia del parser.	xml
.ctor	public Parser()	Crea una instancia del parser.	

Métodos

Método	Declaración	Descripción	Parámetros
Equals	public bool Equals()	Determina si el objeto específico es igual al actual.	obj
Finalize	protected void Finalize()	Permite liberar recursos y realizar otras operaciones de la limpieza antes de que sea reclamado por la colección de la basura	
getAtt	public string getAtt()	Obtiene el atributo de un nodo xml.	node, att
GetHashCode	public int GetHashCode()	Sirve como una función hash para un tipo particular, conveniente para el uso de algoritmos hash y estructuras de datos tales como tablas hash	
getTag	public string getTag()	Obtiene el valor de alguna etiqueta.	element
getType	public string getType()	Obtiene el tipo del mensaje xml.	
getType	public string getType()	Obtiene el tipo del mensaje xml.	element, tag
GetType	public System.Type GetType()	Obtiene el tipo de la instancia actual.	
MemberwiseClone	protected object MemberwiseClone()	Crea una copia bajo el objeto actual.	
ToString	public string ToString()	Regresa una cadena de caracteres que representan al objeto actual.	

A.1.8 Clase *PassivePerception*

Componente de percepción pasiva del agente.

Declaración: *public class PassivePerception : Object*

Campos

Campos	Declaración	Descripción	Parámetros
reasoning	public mSalsa.Reasoning reasoning	Instancia del componente de razonamiento.	

Constructores

Constructor	Declaración	Descripción	Parámetros
.ctor	public PassivePerception()	Asocia la percepción pasiva con un agente.	

Métodos

Método	Declaración	Descripción	Parámetros
Equals	public bool Equals()	Determina si el objeto específico es igual al actual.	obj
Finalize	protected void Finalize()	Permite liberar recursos y realizar otras operaciones de la limpieza antes de que sea reclamado por la colección de la basura	
GetHashCode	public int GetHashCode()	Sirve como una función hash para un tipo particular, conveniente para el uso de algoritmos hash y estructuras de datos tales como tablas hash	
GetType	public System.Type GetType()	Obtiene el tipo de la instancia actual.	
MemberwiseClone	protected object MemberwiseClone()	Crea una copia bajo el objeto actual.	
perceive	public void perceive()	*PassiveEntityToPerceive* notificará la información recibida invocando este método. Cuando un mensaje SALSA es percibido, este método genera un evento SALSA, y notifica a su componente de razonamiento.	input
setReasoning	public void setReasoning()	Asigna el componente de razonamiento al agente.	rsn
ToString	public string ToString()	Regresa una cadena de caracteres que representan al objeto actual.	

A.1.9 Clase *Reasoning*

Componente de razonamiento del agente.

Declaración: *public class Reasoning : Object*

Campos

Campos	Declaración	Descripción	Parámetros
acting	public mSalsa.Acting acting	Componente de acción	

Constructores

Constructor	Declaración	Descripción	Parámetros
.ctor	public Reasoning()	Crea una instancia del componente de razonamiento.	

Métodos

Método	Declaración	Descripción	Parámetros
Equals	public bool Equals()	Determina si el objeto específico es igual al actual.	obj
Finalize	protected void Finalize()	Permite liberar recursos y realizar otras operaciones de la limpieza antes de que sea reclamado por la colección de la basura	
GetHashCode	public int GetHashCode()	Sirve como una función hash para un tipo particular, conveniente para el uso de algoritmos hash y estructuras de datos tales como tablas hash	
GetType	public System.Type GetType()	Obtiene el tipo de la instancia actual.	
MemberwiseClone	protected object MemberwiseClone()	Crea una copia bajo el objeto actual.	
setActing	public void setActing()	Asigna el componente de acción.	acting
think	public void think()	Hay que sobrescribir este método para especificar el algoritmo de razonamiento.	ev, xml
ToString	public string ToString()	Regresa una cadena de caracteres que representan al objeto actual.	

A.1.10 Clase *XMLmessage*

Representa el mensaje xml.

Declaración: *public class XMLmessage : Object*

Campos

Campos	Declaración	Descripción	Parámetros
msg	public string msg	Contiene un mensaje xml.	

Constructores

Constructor	Declaración	Descripción	Parámetros
.ctor	public XMLmessage()	Inicializa la variable msg.	msg

Métodos

Método	Declaración	Descripción	Parámetros
Equals	public bool Equals()	Determina si el objeto específico es igual al actual.	obj
Finalize	protected void Finalize()	Permite liberar recursos y realizar otras operaciones de la limpieza antes de que sea reclamado por la colección de la basura	
GetHashCode	public int GetHashCode()	Sirve como una función hash para un tipo particular, conveniente para el uso de algoritmos hash y estructuras de datos tales como tablas hash	
GetType	public System.Type GetType()	Obtiene el tipo de la instancia actual.	
MemberwiseClone	protected object MemberwiseClone()	Crea una copia bajo el objeto actual.	
ToString	public string ToString()	Regresa una cadena de caracteres que representan al objeto actual.	

A.1.11 Clase *XMLpresence*

Representa un mensaje de presencia.

Declaración: *public class XMLpresence : Object*

Campos

Campos	Declaración	Descripción	Parámetros
presence	public string presence	Contiene un mensaje xml de presencia.	

Constructores

Constructor	Declaración	Descripción	Parámetros
.ctor	public XMLpresence()	Inicializa la variable de presencia.	presence

Métodos

Método	Declaración	Descripción	Parámetros
Equals	public bool Equals()	Determina si el objeto específico es igual al actual.	obj
Finalize	protected void Finalize()	Permite liberar recursos y realizar otras operaciones de la limpieza antes de que sea reclamado por la colección de la basura	
GetHashCode	public int GetHashCode()	Sirve como una función hash para un tipo particular, conveniente para el uso de algoritmos hash y estructuras de datos tales como tablas hash	
GetType	public System.Type GetType()	Obtiene el tipo de la instancia actual.	
MemberwiseClone	protected object MemberwiseClone()	Crea una copia bajo el objeto actual.	
ToString	public string ToString()	Regresa una cadena de caracteres que representan al objeto actual.	

Apéndice B.

Computación ubicua con Calidad de Privacidad (QoP)

En este apéndice presentamos el estudio de privacidad realizado para el componente de migración. Dicho estudio fue desarrollado por [Tentori, 2005] y presenta un esquema de privacidad que puede ser implementado al componente de migración. Dicho estudio es descrito a continuación.

B.1 Introducción

En este capítulo se describe el análisis de los riesgos de privacidad de dos aplicaciones de cómputo ubicuo. Esta evaluación se basa en las guías de diseño descritas en el capítulo IV. Primeramente, se discute el análisis, diseño e implementación de mecanismos de privacidad en un servicio ubicomp y posteriormente, se generaliza en una aplicación de cómputo ubicuo situada en un contexto médico.

El análisis de estas aplicaciones de cómputo ubicuo permite ejemplificar el uso de nuestra propuesta y refinar el diseño de la misma. Elegimos estas aplicaciones por que presentan un conjunto de implicaciones de privacidad en relación a la información que se maneja, las personas con las que se interactúa y la de los mismos individuos durante el uso del ambiente de cómputo ubicuo. Así, podemos evaluar si el uso de nuestra propuesta y un adecuado tratamiento a la información contextual permiten proteger la privacidad de los individuos durante el uso de tecnología de cómputo ubicuo.

El resto del apéndice se encuentra organizado de la siguiente manera: En la sección B.2 se presenta el análisis de un servicio de migración de información. La sección B.3 discute la extensión de una aplicación de comunicación móvil consciente del contexto para hospitales. Finalmente, la sección B.4 presenta la discusión de estos servicios.

B.2 Integrando QoP en un servicio de migración de información

Amaya (Amaya *et al.* 2005), describe un componente de migración que permite *intercambiar y publicar información* entre dispositivos heterogéneos. En un ambiente de cómputo ubicuo existen una gran variedad de dispositivos heterogéneos inmersos en el ambiente con los cuales el usuario puede interactuar. Debido a esta razón, es importante que se faciliten los medios para que el usuario pueda transferir información de un dispositivo a otro fácil y rápidamente. Por ejemplo, enviar información de una PDA a una pantalla pública, y viceversa. Además, cuando un usuario publica o envía información puede perder el control sobre ésta puesto que no se tiene registro quien podría acceder a la información. Sin embargo, sí la información contiene detalles sobre quien tiene derecho a accederla y que privilegios tiene, entonces el pasar la información a un usuario diferente puede ser mantenido bajo control.

El componente de migración descrito en (Amaya *et al.* 2005) funciona de la siguiente manera. La transferencia de archivos inicia cuando el usuario selecciona el archivo que desea transferir, y presiona sobre éste con el botón derecho del *mouse*. En la Figura 68, se observa el resultado de dicha acción. Una vez que el archivo es seleccionado y que aparece el menú de opciones, el usuario debe seleccionar la opción de *Transfer To.* Al seleccionar dicha opción, se despliega una nueva pantalla en la cual aparece el listado de todos aquellos posibles dispositivos destinos a los cuales puede transferirse la información. Cuando el usuario elige el dispositivo destino al cual se desea enviar el archivo, se le solicita a éste si desea aceptar la transferencia. Si la solicitud de transferencia es aceptada por el dispositivo destino, se inicia la migración de la información. Una vez que la información es transferida, se notifica al dispositivo origen que se realizó la transferencia y, en el dispositivo destino se determina la aplicación asociada al tipo de archivo. En el caso de que

el dispositivo destino cuente con la aplicación asociada, se abre la aplicación y se despliega el archivo.

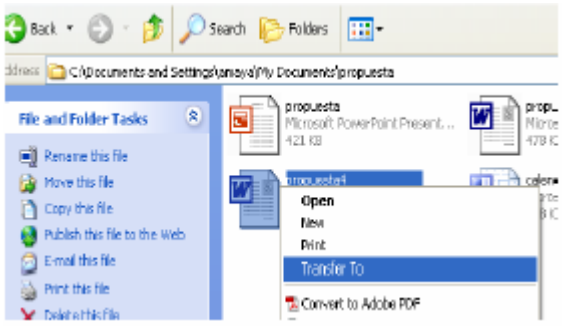


Figura 68. Uso del componente de migración para la transferencia de archivos.

Elegimos este componente como primer caso de estudio por que es sencillo y presenta diferentes riesgos de privacidad ya que el uso y manejo de archivos implica la necesidad de protección de la información. A continuación describimos el análisis de riesgos de privacidad y posteriormente discutimos la manera en que mecanismos de privacidad se pueden incorporar a la aplicación.

B.3 Evaluando el servicio desde una perspectiva de privacidad

Para analizar los riegos y la manera en que se debe de tratar la información contextual relevante para este servicio seguiremos cada una de las fases descritas en el capítulo IV.

B.3.1 Analizando los riesgos de privacidad

Como lo propone la primera fase de las guías de diseño para evaluar los riesgos de la aplicación conducimos un análisis contestando las preguntas en relación a los riesgos que pueden imponer el sensar (sensing), almacenar (storing) y compartir (sharing) información. Para el riesgo de *sensado* la información que se captura es la identidad del individuo o la información de la máquina que funge como emisor. Esta información se captura sin conocimiento del usuario, es decir, cada vez que el usuario utilice el componente este obtiene una referencia automática del individuo, ya sea como un jabberID o como el nombre de la máquina. El hecho de que el usuario no este consciente cuando se captura esta información puede provocar que el usuario no conozca el tratamiento de la

información o se evidencie su identidad. Particularmente para este servicio, esta información no representa una gran amenaza de privacidad ya que únicamente se comparte cada vez que el usuario desde migrar información. Es decir bajo el control del mismo usuario.

Por otro lado el riesgo de *compartiendo,* la información que se comparte son archivos de cualquier tipo, se comparte a cualquier usuario o dispositivo el cual se elige y se transmite bajo consentimiento explícito del usuario que transmite la información. Cuando el receptor recibe el archivo este se despliega automáticamente en la máquina receptora. En este caso se encuentran involucradas dos partes en la interacción el emisor y el receptor. Para el caso del emisor, los riesgos de privacidad son mínimos ya que el usuario tiene el control de la información que comparte; aunque, una vez que se compartió la información el usuario pierde el control de la misma. Por otro lado, para el caso del receptor, los riesgos de privacidad son mayores. Primeramente, el receptor no conoce con exactitud que archivo recibe y quien se lo envía. Además, el hecho que al archivo se despliegue automáticamente puede causar problemas para el emisor evidenciando su información (si la información es altamente confidencial) y para el receptor poniendo en riesgo la integridad de sus archivos (por ejemplo si recibe un virus). En base a esto los riesgos que la aplicación genera son exposición de información, falta de seguridad en el intercambio, pérdida de control en la información compartida y falta de confianza entre los usuarios con los que se comparte la información.

Finalmente en relación al *almacenando,* la información se guarda dentro de un directorio común dentro de la máquina receptora respetando el nombre del archivo con que se inicio la transferencia; acumulando los archivos independientemente de quién inicie la transacción. Esta información se almacena indefinidamente hasta que manualmente se borren los archivos del dispositivo receptor. En base a esto los riesgos que la aplicación pérdida de control del lugar de almacenamiento y no hay garantías del tratamiento de la información a largo plazo.

En resumen, vemos que el componente de migración no tiene mayores implicaciones relacionadas con el riesgo de sensado ya que el componente no monitorea información de los usuarios *per se* únicamente recopila información cuando el usuario realiza la transferencia de archivos. Por otro lado, en lo que se refiere almacenando y compartiéndose pueden generar importantes riesgos de privacidad, por lo que es importante analizar la información contextual que se requiere regular para combatir estos riesgos y la relación entre ellos.

B.3.2 Analizando la información contextual

A continuación se presenta el análisis de la información contextual para evaluar la información contextual relevante que influye en la administración de privacidad durante la migración de información. Después de conducir el análisis encontramos que no todas las variables propuestas son relevantes para proteger la privacidad de este componente y las que importan son identidad, acceso y persistencia.

En relación a la *identidad*, es necesario compartirla ya que el receptor necesita conocer quien envía la información y además se necesitan relacionar los archivos que se transfieren con la persona o dispositivo que envía la información. Mientras no se relacionen los archivos con la persona que envía la información no se tiene riesgo alguno relacionado con la identidad. Pero si se sabe que un archivo pertenece a alguna persona en particular puede ser que esta relación genere la necesidad de proteger la identidad. Es por ello que se deben implementar mecanismos para regular la privacidad de la identidad a dos niveles. El primero de ellos es *anónimo*, en este caso el usuario puede enviar archivos a diferentes dispositivos pero corre el riesgo de que una petición de envío se rechace ya que el receptor puede desconfiar del envío de la información. El segundo nivel es compartir la *identidad real* del individuo, esto incrementa la aceptación de la transferencia del archivo aunque puede quedar un enlace entre el individuo que envía la información y el contenido de la información. Esta necesidad de proteger o de demandar la identidad de una persona depende del tipo de dispositivo y de las personas con las que se interactúe.

En relación al *acceso* la información se comparte para migrar todo el archivo o desplegar información en otro dispositivo. Una vez que el archivo se transfiere es necesario controlar si el archivo se podrá almacenar en el dispositivo y la forma de despliegue (automática o manual). Es por ello que el acceso se debe regular a dos niveles. El primero de ellos es en relación a almacenar la información, en este caso el usuario puede controlar la información por dispositivo si se desea que se almacene o que únicamente se cree una referencia al archivo. Por otro lado, el siguiente nivel esta relacionado con el despliegue de la información, en este caso se puede elegir que automáticamente se despliegue el archivo o que se despliegue hasta que se solicite de manera manual. La necesidad para regular el acceso depende del tipo de dispositivo y de las personas con las que se interactúe.

Finalmente, en relación a la *persistencia* de la información es importante mantener el control del tiempo de vida en que la información puede manipularse para evitar que la información se utilice maliciosamente en otros contextos. Además, no tiene caso almacenar información indefinidamente si no se utilizará con cierta frecuencia. Es decir, si no representa beneficio alguno conservarla, entonces es mejor desecharla. Para regular la persistencia se puede realizar a tres niveles. El primero de ellos es *indefinidamente*, en este caso el usuario podrá acceder la información sin necesidad de realizar constantes transferencias, aunque corre el riesgo de que la información quede expuesta por largos periodos de tiempo. El segundo nivel corresponde a regular la privacidad en función al contexto de *localización*. En este caso la información puede ser relevante únicamente cuando una persona visualiza la información, es decir, no tiene sentido almacenar la información si nadie esta para accederla. Finalmente, el *tiempo* que se almacena la información también es contexto relevante para saber cuando la información se seguirá utilizando.

En resumen, situando el análisis en el servicio de migración encontramos que no existe manera de proteger la privacidad en relación a estos elementos contextuales. Para el caso de *identidad* no existe manera de regularla a diferentes niveles como cuando el usuario no desea que se mantenga una referencia entre el archivo que comparte y su identidad. Por otro lado, no existe manera de controlar las acciones que se pueden realizar con la

información que se comparte, por ejemplo cuando el dueño de un archivo desea que esta información no se despliegue automáticamente. Finalmente, no se controla el tiempo de vida de la información, como cuando el dueño de la información desea que solo este disponible el periodo de tiempo que dura la sesión. Aunque a través de este análisis podemos identificar la información contextual relevante para regular la privacidad, no tiene sentido administrarla, si en instancias específicas no representa una amenaza para los usuarios potenciales.

B.3.3 Diseñando escenarios de privacidad

Una vez que se tiene un claro entendimiento de los riegos de privacidad y los elementos contextuales relevantes, es necesario evaluar estos resultados en un contexto específico para ver si estas amenazas cobran sentido en la práctica. A continuación se describe uno de los escenarios que enfatizan los riesgos de privacidad.

Un paciente del hospital presenta una evolución y reacción extraña al tratamiento por lo que médicos internos, médicos de base y algunos médicos especialistas discuten el caso clínico en conjunto. Mientras están en la sesión clínica utilizan una pantalla pública para consultar el expediente del paciente y los resultados de los estudios que se realizaron. Mientras discuten la información algunos médicos comparten artículos de referencia y notas personales relevantes a la discusión para enriquecerla con evidencia médica (ver Figura 69). Juan, un médico de base, tiene un artículo y una presentación relevantes al caso. Juan desea discutir el artículo y en caso necesario ejemplificarlo con una presentación, por lo que desea que el artículo se despliegue automáticamente pero no así la presentación. Además, desea restringir el acceso a estos archivos permitiendo su uso únicamente durante la sesión. Por lo que requiere que esta información se borre de la pantalla pública a donde migro esta información cuando la sesión finalice.

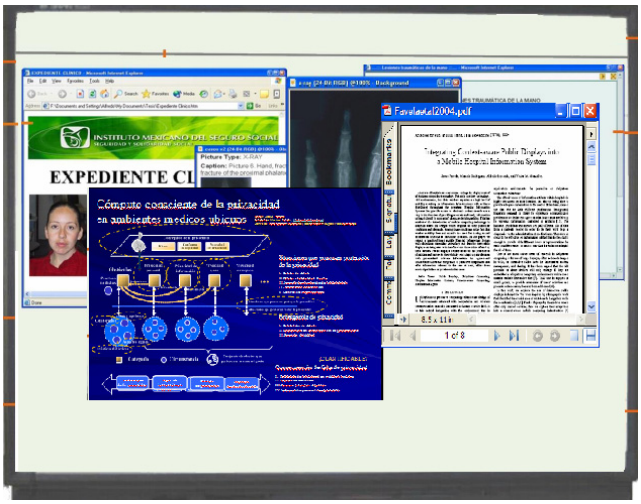


Figura 69. Migración de archivos desde dispositivos personales a una pantalla pública.

Como muestra el escenario el médico desea controlar el acceso y la persistencia de la información que se transfiere a la pantalla pública. Para el caso de *acceso* Juan no desea que la presentación se despliegue automáticamente, pero el artículo. Por otra parte, no desea que esta información se almacene por un periodo largo de tiempo; únicamente desea que este disponible por el tiempo que dura la sesión. En este caso es difícil saber con exactitud el tiempo que dura una sesión, ya que muchas veces se pueden prolongar o durar menos del tiempo que se programe. Es por ello, que será más relevante administrar la persistencia en función de la localización de Juan, es decir, cuando Juan se aleje de la pantalla pública donde esta el archivo es necesario borrar la información.

En base a esta información se puede observar que la identidad, acceso y persistencia se pueden regular a través del uso de una escala de valores para estos elementos. Pero es necesario identificar las soluciones que ayudan a regular los riesgos que presentan.

B.3.4 Integrando consciencia de privacidad

Para esta fase construimos la matriz solución para los elementos de identidad, acceso y persistencia relacionados con el componente de migración. La Tabla VI muestra la matriz solución con los mecanismos de privacidad que necesitan incorporarse al componente de migración para administrar la privacidad.

TABLA VI. Discusión de archivos proyectados por la audiencia permitiendo el envío de mensajes.

Soluciones	Identidad	Acceso	Persistencia
INFORMANDO			
Notificación	√	√	
Retroalimentación	√	√	
Acceso			
APRENDIENDO			
Control		√	√
Elección y consentimiento			
REACCIONANDO			
Reacción ante el contexto		√	√
Administración de precisión	√		

Cuatro mecanismos se eligieron para regular la privacidad para el componente de migración: notificación, retroalimentación, control en base al contexto y administración de la precisión. El mecanismo de *notificación* se utiliza para avisarle al usuario de la persona que envía información, si desea que la información se salve en el dispositivo al que se transfiere o si desea que la información se despliegue automáticamente. Este mecanismo se utiliza en caso de que el usuario no tenga especificadas políticas de control para el archivo que transfiere a otro dispositivo. Además, de este mecanismo se implementará la solución de *retroalimentación* la cual permite mantener informado al usuario acerca de la información de identidad que comparte con el sistema, las acciones que se pueden realizar sobre un archivo en particular (almacenamiento o no, …), la persona que envía el archivo y la persistencia del archivo. El siguiente mecanismo que se eligió para proteger el acceso y la persistencia fue el de *control*. Este mecanismo permite especificar políticas de privacidad de manera explícita (por medio de archivos de configuración) o implícita en

(función al contexto). Finalmente, es importante agregar un mecanismo para que el usuario pueda cambiar la precisión con la que comparte su información de identidad.

Con estos cuatro mecanismos se logra mantener informado al usuario, se le delega el control y se minimiza el esfuerzo que realiza en la administración de privacidad por lo que esto es suficiente para proveer consciencia de privacidad. Los otros mecanismos que no se implementan aunque ayudan a proteger la privacidad complicarían más el diseño. Una vez que se tiene los mecanismos de privacidad a incluir, es necesario encontrar una escala de valores adecuada para regular los elementos de información y así, proteger la privacidad en el uso del sistema. Para ello creamos una ontología y diferentes políticas por omisión para regular los elementos contextuales relevantes para este servicio.

B.3.5 Clasificando y nivelando la información contextual

La Tabla VII muestra la ontología que se generó para el uso del servicio de migración la cual muestra los elementos de información contextual necesarios para regular la privacidad de este servicio (identidad, acceso y persistencia) y los valores de cada uno de ellos, que se derivan del mapeo de la información contextual y los escenarios de uso en un contexto específico. La privacidad que demande un usuario va a depender de la combinación de estos valores a diferentes escalas dependiendo de la situación en que se encuentre el usuario.

TABLA VII. Ontología para el servicio de migración.

Niveles	Identidad/Receptor	Acceso	Persistencia
1	Nombre	Total	Indefinida
2	Anónimo	Despliegue automático	Por localización
3		Almacenamiento	Por tiempo
4		Despliegue	

En base a los valores que los elementos contextuales pueden tomar se pueden crear diferentes combinaciones para administrar la privacidad y estipular configuraciones por omisión. En este caso el sistema automáticamente proveerá cuatro tipos de configuración predefinidas, las cuales difieren en la restricción de privacidad para cada uno de los elementos contextuales. La primera de ellas NDI corresponde a la combinación de identidad como nombre, acceso como almacenamiento con despliegue automático y persistencia indefinida. La segunda configuración NSL corresponde a compartir la identidad como nombre, acceso de almacenamiento sin despligue automático y persistencia de información en base a la localización. Finalmente la tercer configuración ADT corresponde a compartir la identidad como anónimo, acceso únicamente de despligue y persistencia en base al tiempo.

Este sistema aunque cuenta con pocos riesgos en relación a la privacidad personal cuenta con muchos en relación a al privacidad de la información. Esto nos ayuda a evaluar si nuestras propuestas apoyan las tres dimensiones que describen la privacidad en la práctica diaria (privacidad personal, privacidad de la información y privacidad ajena).

A través del análisis identificamos que *no existe manera de proteger la identidad del usuario que comparte la información, no se controla el acceso a la información que se comparte entre el receptor y las operaciones que se pueden realizar con la información y finalmente, no se controla el tiempo de vida de la información que se comparte.* Estos riesgos en caso de presentarse generan las siguientes consecuencias: *evidencia de la identidad personal, desconocimiento de la manera en que se utilizará la información recopilada, exposición de información, pérdida de control en la información compartida y falta de confianza entre los usuarios con los que se comparte la información.* Además, se analizaron los riesgos en diferentes escenarios de uso específico para los elementos de información de identidad, acceso y persistencia obteniendo los posibles valores para regular la privacidad. Para lidiar con esto nosotros extendimos la arquitectura del componente de migración agregando la capa de privacidad propuesta en nuestra arquitectura, generando la ontología y agregando mecanismos de privacidad al diseño de

este servicio. A continuación se ejemplifica como estos mecanismos se incorporaron a la aplicación.

B.3.6 Incorporando condiciones de privacidad

Una vez que concluimos el análisis de riesgos y el análisis de soluciones decidimos implementar los mecanismos que se encontraron en la matriz solución de la siguiente manera. Para manejar la identidad se tomará automáticamente el identificador de la máquina en la que el usuario haya iniciado sesión y se provee un mecanismo para que el usuario pueda regular su identidad cambiándola como anónimo (ver Figura 70a).

Figura 70. El usuario a través de una PDA puede regular el nivel de privacidad para su identidad (a), recibir retroalimentación por medio de la barra de información (a) o mensajes interactivos para modificar sus condiciones de privacidad (b), establecer las acciones que pueden realizarse con la información (c) y administrar sus políticas de privacidad (d)

En relación al acceso y persistencia se proveen dos soluciones, la primera de ellas para mantener al usuario informado y la segunda para permitir que el usuario controle su información. Para mantener al usuario informado, se implementó el mecanismo de *retroalimentación* a través de una barra que cambia según el contexto del usuario (ver figura 70a), y además, se implementó el mecanismo de *notificación*, para interactuar con el usuario y adaptar el comportamiento del sistema en base a los requerimientos de privacidad del usuario. Por otro lado, en relación al control se provee una manera para que el usuario cree diferentes combinaciones para administrar el acceso (ver figura 70c) y la persistencia de la información. Finalmente, el estado de la configuración se puede almacenar para

reducir la necesidad de configurar el nivel de privacidad posterior y seleccionar una política de privacidad durante el inicio de sesión (ver figura 70d).

B.4 Extendiendo la arquitectura del componente de migración

La figura 66 muestra la arquitectura del componente de migración descrito en (Amaya *et al.* 2005), la cual, implementa los filtros de privacidad del lado del cliente y del servidor. Como muestra la figura 71 los filtros de privacidad se agregaron como Proxies de lado del cliente y del lado del servidor. Esto quiere decir que cada petición entre el cliente contextual y el servidor contextual, será filtrada por estos proxies. A continuación se explica el funcionamiento general de los filtros agregados.

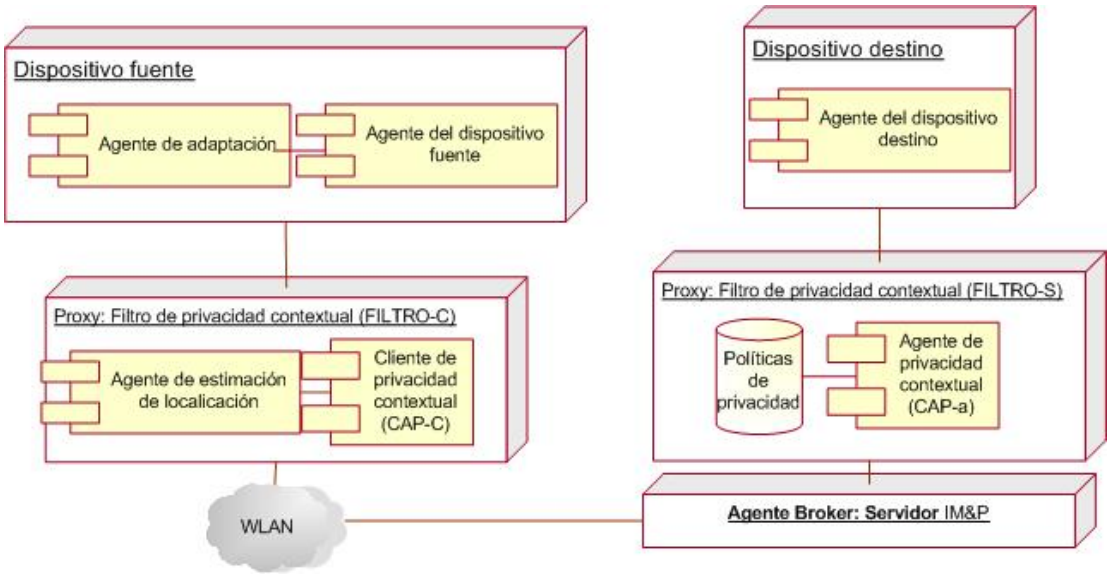


Figura 71. Arquitectura extendida integrando consciencia de privacidad y los mecanismos para sensar contextualmente la localización de los individuos

B.4.1 Filtro de privacidad contextual del lado del cliente

El acceso a la información y la posibilidad de interactuar con el medio se hace a través de un dispositivo móvil (PDA), de una pantalla pública (public display) o de una computadora de escritorio. El cliente de privacidad contextual mostrado en la figura 72 utiliza el módulo "cliente Jabber" para comunicarse con el servidor de privacidad contextual y con los demás usuarios que forman parte del espacio donde se encuentra el usuarios.

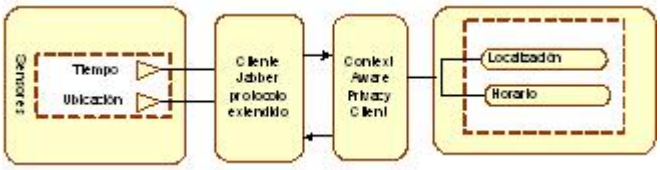


Figura 72. Cliente de privacidad contextual

El módulo "elementos contextuales" contiene las variables contextuales que definen el contexto del usuario en un determinado momento. Este modulo permite al usuario definir el contexto en el que sus elementos de información se deben adaptar de acuerdo a cierto nivel de QoP. El módulo "sensores contextuales" permite obtener la información del contexto del usuario para deducir cuando la privacidad se debe regular.

B.4.2 Filtro de privacidad contextual del lado del servidor

El ambiente esta representado por el servidor de privacidad contextual (ver figura 73), el cual mantiene registrada la información de las políticas de privacidad (repositorio de políticas), la información de comportamiento de acceso (repositorio de acceso) y finalmente la información contextual que caracteriza al usuario en el ambiente. Esta información se envía a cada uno de los usuarios que ingresan dentro del área de cobertura del servidor, analizando los conflictos con las políticas estipuladas y adaptando el comportamiento de los elementos de información. El servidor conciente del contexto utiliza el módulo "cliente Jabber" para comunicarse con los elementos que forman parte del ambiente que representa. Utiliza el módulo "activador de eventos" para analizar el contexto y reaccionar de acuerdo al mismo.

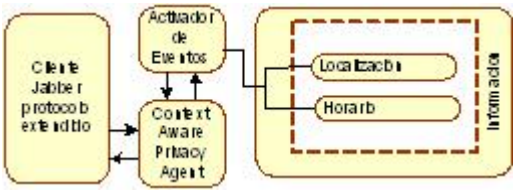


Figura 73. Servidor de privacidad contextual

B.4.3 Ejemplo de aplicación

Para ejemplificar el uso de nuestra solución expondremos aquí la interacción de los filtros con los agentes que pertenecen a la arquitectura del componente de migración. Del escenario descrito anteriormente se ejemplificará como una vez que los mecanismos de privacidad se implementan el usuario puede controlar el despliegue automático o manual de un archivo. En este caso utilizando los componentes agregados para la administración de la privacidad el usuario puede decidir el tratamiento adecuado a la información que comparte con la pantalla pública. En el diagrama de secuencia de la figura 74 se muestra la secuencia de actividades y la interacción de los componentes de la aplicación.

El usuario visualiza en su PDA una interfase que le muestra la forma para transferir archivos. El usuario selecciona la presentación que desea mostrar en el pizarrón electrónico sin almacenarla por mucho tiempo y sin que se despliegue automáticamente. El usuario selecciona el nivel de privacidad eligiendo la política NSL. El usuario y filtro de privacidad contextual del lado del cliente negocian a través de este archivo de políticas de privacidad el nivel de privacidad del usuario. El sistema inicia la sesión con el servidor jabber. El usuario presiona el botón de transferencia de archivos y el sistema anuncia el nivel de privacidad seleccionado por el usuario. Además, al seleccionar dicha opción, se despliega una nueva pantalla en la cual aparece el listado de todos aquellos posibles dispositivos destino a los cuales puede transferirse la información.

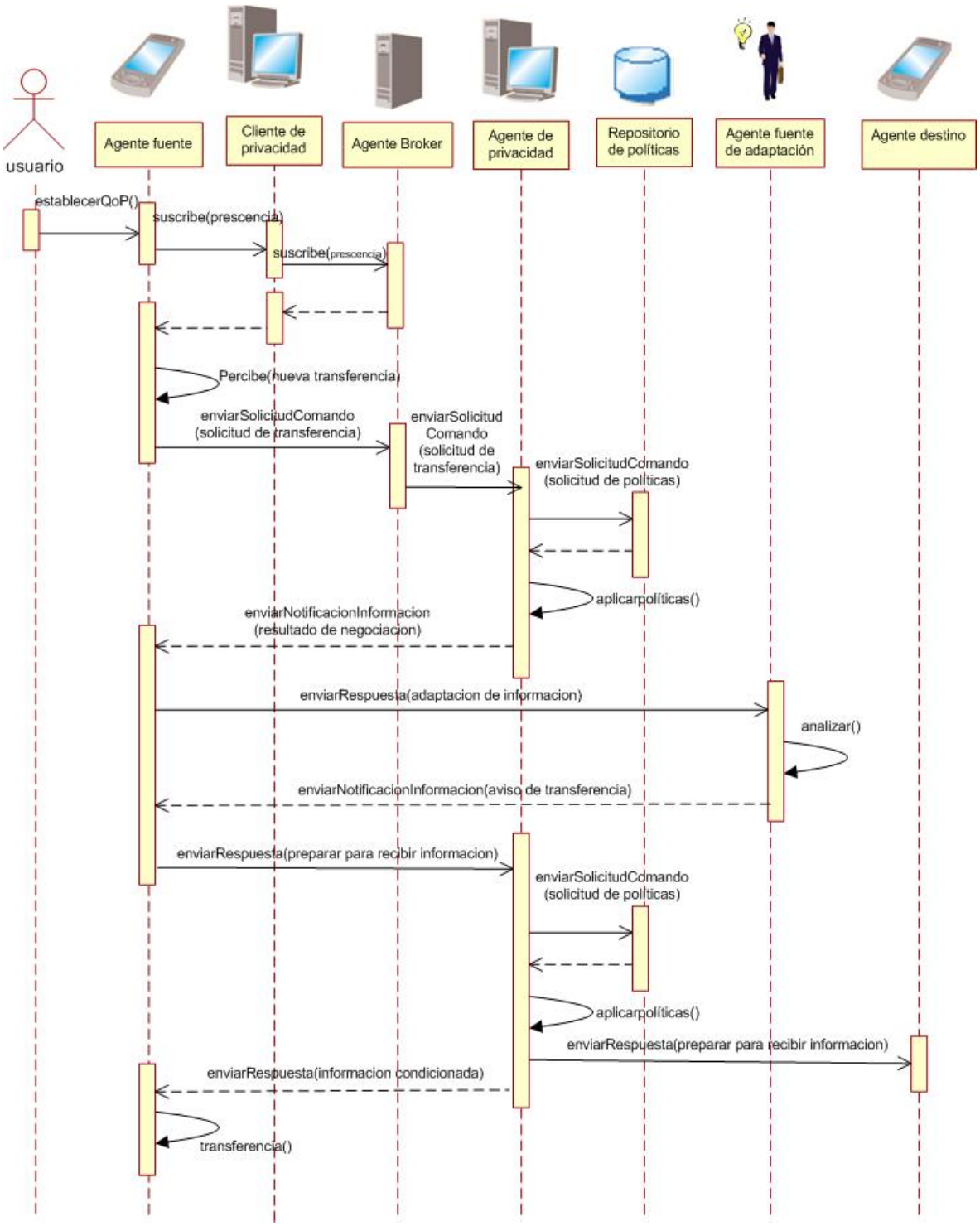


Figura 74. Diagrama de secuencia que muestra el uso de condiciones de privacidad para la transferencia de información sin despliegue

El usuario selecciona la pantalla pública. El filtro de privacidad del lado del servidor consulta las políticas de la pantalla pública y las compara con las políticas del usuario. El nivel de privacidad de ambos concuerda por lo que la transferencia se acepta. La transferencia de información se inicia entre la PDA del usuario y la pantalla pública. Una vez que la información se transfiere se revisan las políticas de privacidad para estipular las acciones que se pueden realizar con el archivo. En este caso la calidad de privacidad de despliegue de la pantalla pública entra en conflicto con la calidad de privacidad demandada por el usuario, por lo que se deniega el despliegue automático del archivo.